ערכת כלים

Visual Basic 6



יוסי שריקי

✓ **למתכנת המתחיל**
כלי פיתוח ויזואליים

✓ **למתכנת שפיתח**
בגרסאות קודמות

✓ **תרגול, תרגול, תרגול**







יוסי שריקי

6 Visual Basic

ערכת כלים



Visual Basic 6 ערכת כלים

יוסי שריקי

שפת Visual Basic הינה שפה קלה ללימוד שאפשר לעשות איתה כמעט הכל. אך אל תניח לעושרה ולפשטותה להטעות אותך. בנוסף ללימוד התחביר אתה חייב ללמוד גם הרגלי חשיבה טובים ולתרגל.

ספר זה יקנה לך יסודות מוצקים בעבודה עם Visual Basic החל מהכרה בסיסית ועד לתכנות נושאים מורכבים. הספר מלווה בתרגילים שיעזרו לך לבחון את מה שלמדת, ולהוכיח כי אתה יכול ליישם זאת.

המחבר **יוסי שריקי**, מפתח ב- Visual Basic בעל ניסיון רב, אחראי על צוות פיתוח ומרצה במרכז הדרכה סיון.

הספר מיועד:

- ✓ ללומדים תכנות שזוהי שפת התכנות הראשונה שלהם.
- ✓ למתכנתים בעלי ניסיון בעבודה עם Visual Basic בגרסאות קודמות הרוצים להתעדכן ביכולות של גירסה 6 וללמוד דברים נוספים.

התקליטור המצורף כולל:

- ✓ Visual Basic 6 בגירסה מיוחדת.
- ✓ קוד מקור.
- ✓ תוכנות עזר.
- ✓ בונוסים.

רמת משתמש
מתחיל | בינוני | מתקדם

מחיר מומלץ לצרכן 119 ש"ח



מסת"ב ISBN 965-361-185-2

Visual Basic 6
ערכת כלים

הוראות התקנה והפעלה של התקליטור המצורף - בנספח ובקובץ
ONCD שבתקליטור

עורך ראשי: **יצחק עמיהוד**

עיצוב ועריכה לשונית: **שרה עמיהוד**

עיצוב עטיפה: **שרון רז**

תודה ליהודה כץ על הערותיו

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי עשתה כמיטב יכולתה למסור מידע אודות השמות המסחריים המוזכרים בספר זה ולציין את שמות החברות, המוצרים והשירותים. שמות מסחריים רשומים (registered trademarks) המוזכרים בספר צוינו בהתאמה.

Windows הינו מוצר רשום של חברת Microsoft

Visual Basic הינו מוצר רשום של חברת Microsoft

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716 1-700-7000-44**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

Visual Basic 6
ערכת כלים

יוסי שריקי





Visual Basic 6 ToolKit

By **Yossy Shriki**

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון: 09-9564716 פקס: 09-9571582

info@hod-ami.co.il

הודפס בישראל 1999, 2000, 2001, 2002

הדפסה 10/2003

ISBN 965-361-185-2 מסת"ב

תוכן העניינים

17: הוספת סרגל כלים ליישום 299

18: דוחות 311

הקדמה

בעידן המחשבים של ימינו נמצא כמעט בכל בית מחשב עם מערכת הפעלה Windows. כך גם בכל עסק, קטן ככל שיהיה, במפעלים גדולים וחברות, מכיון שהפעילות העסקית נעזרת כיום יותר במערכות ממוחשבות. מערכות עסקיות וביתיות שפעלו בסביבות עבודה קודמות כבר הוסבו, או שהן מוסבות כעת למערכת הפעלה Windows 95 ובעצם כבר למערכת ההפעלה החדשה Windows 98. הדבר מתבטא בקורסי ההסבה הרבים המועברים למזכירות, למנהלים ולכל אדם אשר במסגרת תפקידו זקוק למחשב. מציאות זו צריכה להעמיד אותנו – המתכנתים – עירניים למצב המשתנה והמתחדש, להכיר וללמוד את התוכנות ולתכנת בהן ברמה גבוהה ומקצועית.

לאור הצורך הגדול בשפה חלונאית, ובפרט לסביבת Windows 95 וסביבת Windows 98, דרושה שפה בעלת עוצמה וכלים לפיתוח יישומים ברמה גבוהה, שתיתן פתרונות מקצועיים למיגוון הרחב של המשתמשים הזקוקים לשירותי מיחשוב. Visual Basic (בהמשך, נשתמש בכיתוב עברי : ויז׳ואל בייסיק) הינה שפה העונה על דרישה זו. ויז׳ואל בייסיק היא שפת תכנות ברמה מקצועית לסביבת Windows המאפשרת פתרונות יישומיים ברמה גבוהה בסביבת העבודה של המחשבים האישיים. חברת Microsoft פיתחה בעזרת ויז׳ואל בייסיק רבות מהתוכנות שלה, או חלק מהשירותים שהן מספקות. יתרון נוסף של ויז׳ואל בייסיק הוא פשטות השפה ולכן – קלות הלימוד והיישום. על אף מיגוון האפשרויות העצום של השפה, היא קלה ללימוד ופשוטה לשימוש. מי שמכיר את שפת Basic הישנה על מיגוון גרסאותיה, ימצא חידושים מרעננים ומתקדמים.

ספר זה מיועד למתכנתים בשפת Visual Basic גירסה 6. הוא יאפשר למתכנתים מתחילים, ללא ניסיון כלל בוויז׳ואל בייסיק, ללמוד ולהתקדם שלב אחר שלב, מבסיס השפה ועד לרמה מקצועית גבוהה.

גם מתכנתים בעלי ניסיון בשפה, יוכלו להפיק תועלת מהספר, החל בפרקים העוסקים בנושאיה המתקדמים של השפה, אשר לא הכירו קודם.

מסיבה זו חילקתי את הספר לשני חלקים :

החלק הראשון כולל צעדים ראשונים, ומכוון למי שאינו מכיר את השפה. בצעדים קטנים הוא יוכל להתקדם מהכרה בסיסית ועד לשליטה באפשרויותיה הרבות.

החלק השני הוא הדרכה לתכנות מקצועי, ומבוסס על הבסיס שניתן ללומד בחלק הראשון של הספר. הוא מתאים גם לקורא שיש לו רקע קודם בגירסה קודמת של השפה. מטרתו להביא את הקורא לרמת מתכנת ויז׳ואל בייסיק מקצועי מן השורה. מתכנתים בעלי ניסיון בוויז׳ואל בייסיק יוכלו לפנות ישירות לחלק השני, שבו הם ימצאו חומר מתקדם וגם המלצות לתכנות יעיל ונכון.

הספר כתוב בשיטת צעד-אחר-צעד, או שיטת השלבים, שבה כל פרק מבוסס על החומר שנלמד בקודמו. מומלץ מאוד לקרוא וללמוד את הפרקים לפי סדר כתיבתם. קריאה בלבד אינה מאפשרת להפיק את מירב התועלת מן הלימוד. כדי לקצור פירות חייבים לעמול, והכוונה – לתרגל! התרגילים אינם המלצה בלבד כי אם חלק מהלימוד. הרוצה להפיק מירב מן הספר לא יפסח על אף תרגיל. לא זו בלבד שהתרגילים מקנים ניסיון והרגלי חשיבה והרגלי תכנות טובים, אלא שלרוב אפשר ללמוד בעזרתם דברים נוספים לאלה שנלמדו במסגרת החומר התיאורטי.

במהלך הלימוד בספר השתמשתי במונחים המקצועיים המקובלים, וגם במונחים המתורגמים לעברית. כך יכול הלומד להעשיר את יכולתו גם לקריאה בספרות המקצועית באנגלית.

ספר זה מבוסס על שתי גרסאות של Visual Basic 6: המהדורה המפעלית (**Enterprise**) והמהדורה המקצועית (**Professional**). בחלק מהנושאים הוא מתאים גם לגירסה הסטנדרטית (**Standard**), אך לא נעשתה בדיקה של התוכניות בגירסה זו.

בתקליטור המצורף לספר זה תמצא את גרסת **Visual Basic 6 - Working Model**. גירסה זו של התוכנה ניתנת לך כבונוס ואין הספר מתיימר ללמד אודותיה. כמו כן, קוד המקור בספר זה **לא** נבחן על גירסה זו (הוראות ההתקנה בתקליטור ובנספח).

בתקליטור תמצא את קוד המקור של מרבית קטעי הקוד המוזכרים בספר. ראה הסבר בנספח בסוף הספר ובקובץ ONCD בתקליטור.

לא נותר, אלא לאחל לך הקורא-לומד דרך צלחה, עבודה קשה וקריאה נעימה.

מבוא

תהליך פיתוח של יישום ויז'ואל בייסיק במחשב, חייב לכלול ארבעה שלבים, ללא תלות בגודלו. אנו מניחים ששלב התכנון המוקדם הוא ברור מאליו, ונעשה עוד לפני שהמתכנת הדליק את המחשב.

לאחר שבירררנו לעצמנו מהי התוכנה שאנו עומדים לפתח, ופרטיה מפורטים ומתוכננים על הנייר, כל שנשאר הוא לתרגם את אשר נכתב לשפת תכנות ובמקרה שלנו - Visual Basic. ארבעת השלבים הם:

א. יצירת ממשק משתמש.

ב. קביעת מאפיינים.

ג. כתיבת הקוד.

ד. יצירת קובץ הרצה (וערכת התקנה).

בניגוד לשפות פרוצדורליות, אנו מתחילים ביצירת ממשק משתמש עוד לפני שכתבנו אפילו שורת קוד אחת. אנו מעצבים את המסך - או במונח המקצועי את **הטופס** (Form) כפי שייראה למשתמש בסוף התהליך. עיצוב הטופס נעשה באמצעות "פיזור" **פקדים** (Controls - הם כלי העבודה) על פניו. לדוגמה, נחליט כי התוכנית שלנו תציג הודעה ותאפשר קלט של מספר מהמשתמש. אחר כך הוא ילחץ על הלחצן, או פקד הפקודה, ויקבל כתוצאה את החזקה הריבועית של המספר שהקיש. לצורך כך ניצור טופס כמודגם בתרשים:



תרשים: טופס לדוגמה

על פני הטופס מוצגים שלושה פקדים: **פקד תווית** (Label) שבו נכתבה ההוראה המנחה "הכנס מספר:". פקד **תיבת טקסט** (TextBox) מאפשר למשתמש להקליד את המספר שבחר. הפקד **לחצן פקודה**, ובקיצור - **לחצן** (CommandButton), בו רשום "העלאה בחזקה", ואשר בלחיצה עליו נקבל את התוצאה הרצויה.

בשלב הבא עלינו לקבוע את המאפיינים. לכל פקד נקבעים המאפיינים המתאימים לו.

בדוגמה שלנו, המאפיין **Caption** (כותרת) של הפקד Label הוא "הכנס מספר:". במקום לומר כי על התווית נכתבה ההוראה "הכנס מספר:", נאמר בשפה מקצועית כי

המאפיין Caption של הפקד Label הוא "הכנס מספר:" (או, מכיל את הערך "הכנס מספר:").

היכן נקבע כי המאפיין Caption של הפקד Label הוא "הכנס מספר:" ? נראה בהמשך כי יש חלון מיוחד, **חלון המאפיינים** (Properties Window), בו רשומים המאפיינים של כל פקד. בחלון זה נקבע ערכו של כל מאפיין.

על חלון המאפיינים לא נרחיב כאן את הדיבור, נפגוש אותו בפרק הראשון ובמהלך לימוד התוכניות בפרקים שלאחר מכן. ערך המאפיין שקיבל הפקד בראשונה אינו בהכרח קבוע לכל אורך התוכנית; הוא ניתן לשינוי כפי הצורך. השינוי לא ייעשה דרך חלון המאפיינים, כי אם באופן דינמי באמצעות שורות קוד בלבד.

בשלב השלישי כותבים את הקוד. אין זה שלב הכרחי בכל יישום, ויש יישומים הפועלים ללא שורות קוד כלשהן. יישומים אלה הם בדרך כלל פשוטים ומוגבלים ביכולתם. מכיון שמטרתנו לפתח יישומים ברמה מקצועית, נוסיף כתיבת קוד כפי הצורך.

הזכרנו קודם שבאפשרותנו לשלוט בערכי מאפייני הפקדים באמצעות שורות קוד. אולם, נוח יותר לקבוע ערכים ראשוניים למאפיינים באמצעות חלון המאפיינים. באמצעות שורות הקוד נבצע את השינוי הדינמי של המאפיינים, אך אפשר לקבוע ערך באמצעותן גם בפעם הראשונה.

קביעת ערכו של מאפיין בעזרת שורת הקוד תיעשה באופן הבא:

```
Label.Caption = "הכנס מספר:"
```

תחילה נציין את שם הפקד, אחריו נקודה, שם המאפיין, אופרטור השוויון "=" (שווה) ולבסוף הערך הרצוי.

בחלק זה באה לביטוי שפת Basic "משופרת" (אשר דומה מאוד לזו שהכרנו בעבר), וככל שפה גם לה תחביר (Syntax) המאפשר למתכנת לתכנת על פי כללים מוסכמים.

בשפת ויז'ואל בייסיק אין משמעות לכתיבה באותיות קטנות או גדולות, ואין הבדל בין שני סוגי האותיות לפיכך, Sort זהה ל-sort ו-CheckInventory זהה ל-checkinventory. התוכנה עצמה צובעת באופן אוטומטי בכחול ומציינת באות רישית את המילים השמורות לשפה.

כאן המקום לציין את **תיעוד** שורות הקוד. מיותר להדגיש את חשיבות התיעוד של עבודת התכנות, אשר מתבטא בעיקר בכתיבת הערות בין ובהמשך לשורות הקוד. נאמר רק, כי הדרך לתעד היא בעזרת התו גרש - **'** - (הגרש נמצא במקש של האות W). ההערה נכתבת בשורה נפרדת, או בשורות נפרדות, וגם אפשר להוסיף אותה בסוף שורת הקוד. לא ניתן לשלב הערה באמצע שורת קוד! את ההערה ניתן לכתוב באנגלית או בעברית.

הדוגמה הבאה ממחישה כיצד להוסיף הערה לשורות הקוד שלנו:

```
'Loop in Visual Basic 6
For i = 1 To 10            'Looping ten times
   Label.Caption = i
Next i
```

השלב האחרון בפיתוח היישום הוא יצירת קובץ הרצה (exe), אשר יפעל ללא צורך בתוכנת ויז'ואל בייסיק. לצורך זה יש לבצע תחילה הידור ולוודא שאין שגיאות בתוכנית. לאחר שעוברים גם שלב זה בהצלחה, כל שנותר לעשות הוא ליצור את קובץ ההרצה (exe). אך עדיין לא ניתן למסור את היישום למישהו אחר להפעלה. למרות שיש לנו קובץ הרצה, הוא קשור לקבצי DLL שונים והוא אינו יכול לפעול בלעדיהם. כדי להפעיל את היישום שפיתחנו גם במחשבים שאין בהם ויז'ואל בייסיק, נצטרך להכין ערכת התקנה מתאימה. לצורך בניית ערכת ההתקנה ניעזר בתוכנית Setup Wizard, שבה נעסוק בפרק החותם את הספר.

כאן נסיים את המבוא ונאחל לך לימוד מהנה ומועיל.

הספר מוקדש באהבה

לאשתי יפעת

ולילדיי

אראל שמעון

איתן רפאל דוד

הלל

יוסי שריקי

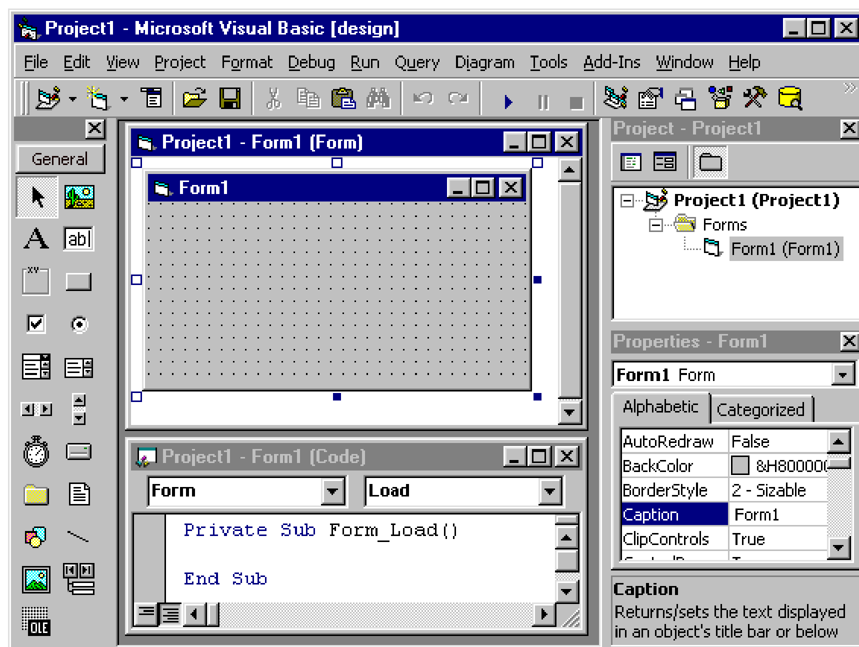
חלק א'

צעדים ראשונים

1: מרכיבי המסך

בעבודתנו עם ויז׳ואל בייסיק אנו פועלים בשני מצבים. האחד, **זמן עיצוב** או **זמן פיתוח** (**Design Time**), בו אנו נמצאים ברוב זמן עבודת הפיתוח שלנו. המצב השני הוא **זמן פעולה**, **או זמן ריצה** (**Run Time**), הוא הזמן בו התוכנית שלנו פועלת ומבצעת דברים. במצב זה נמצא המשתמש בעת עבודתו.

בפרק זה נציג את החלקים השונים המרכיבים את המסך בזמן פיתוח. נלמד להכיר את המרכיבים האלה באופן כללי, להציג אותם ולהסתירם בעת הצורך, וגם נכיר את תפקיד כל אחד מהם.



תרשים 1.1: מרכיבי המסך

שורת התפריטים (MenuBar)

כמו בכל תוכנה חלונאית, כך נמצא גם בוויז׳ואל בייסיק את שורת התפריטים. בשלב זה של הלימוד לא נעסוק בכל האפשרויות שנמצאות בתפריטים. נעסוק כעת באפשרויות השימושיות ביותר, ובאחרות נרחיב את הדיבור במהלך הלימוד.

נתחיל בתפריט **File**. בתפריט זה נמצא את האפשרויות השכיחות האלו:

New Project - ליצירת פרויקט חדש. בבחירת אפשרות זו נקבל מבחר סוגי פרויקטים; נבחר באפשרות הראשונה המוצעת - **StandardEXE** . זהו פרויקט סטנדרטי למטרת יצירת קובץ הרצה.

Open Project - לפתיחת פרויקט קיים. תיבת דו-שיח תאפשר בחירה של הקובץ שאנו רוצים לפתוח. הקובץ שנבחר יאופיין בסיומת **vbp** (Visual Basic Project) הנהוגה עבור קבצי ויזואל בייסיק (ההבחנה בין קובץ פרויקט לבין קבצים אחרים המשתתפים ביישום תידון בהמשך הפרק).

Save Project - לשמירת הפרויקט. התוכנה אינה מבצעת שמירה אוטומטית, ולכן מומלץ לשמור את הפרויקט לעיתים קרובות.

תפריט **Edit** מכיל את כלי העריכה המוכרים בסביבת Windows ובהם:

Copy - להעתקת פקד או קטע קוד נבחר.

Cut - לגזירת פקד או קטע קוד נבחר.

Paste - להדבקת פקד או קטע קוד נבחר.

מתכנתים מקצועיים ממעטים להשתמש בתפריטים, ונעזרים בעיקר במקשי הקיצור (shortcut). השימוש במקלדת הופך את עבודת המתכנת לקלה, נוחה ומהירה. מומלץ מאוד להקנות הרגלי עבודה אלה כבר בהתחלה, ואת הפירות נקצור בהמשך.

גם הלחצנים העומדים לרשותנו בסרגל הכלים ממלאים פונקציה דומה למקשי הקיצור. ראוי ורצוי ללמוד את תפקידם, ובעיקר של אלה השימושיים יותר. אזכיר כי, כמקובל בסביבת Windows, הנחת סמן העכבר על פני הלחצן לשנייה או שתיים, תגרום להופעת תיאור כלי (Tooltip) שמסביר את תפקיד הלחצן.

בתפריט **Run** נמצא את האפשרויות הבאות:

Start - להרצת הפרויקט (העברתו מזמן פיתוח לזמן ריצה). עדיין אין הופכים את הפרויקט לקובץ הרצה.

End - סיום זמן ריצה וחזרה לסביבת הפיתוח.

בכל הנוגע ל**עזרה**, מערכת העזרה בגירסה 6 שונה ממה שהכרת בגירסה 5.

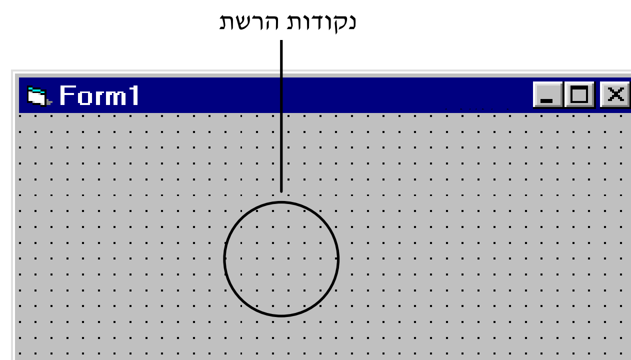
כדי לקבל עזרה, בזמן ההתקנה של Visual Basic מתוך ה-Visual Studio 6 עליך להתקין גם את **MSDN** שזו מערכת העזרה. תוכל להתקין את MSDN גם לאחר שהתקנת את התוכנה עצמה.

Contents - בה תמצא את מערכת העזרה של ויז'ואל בייסיק (אם מותקן MSDN).

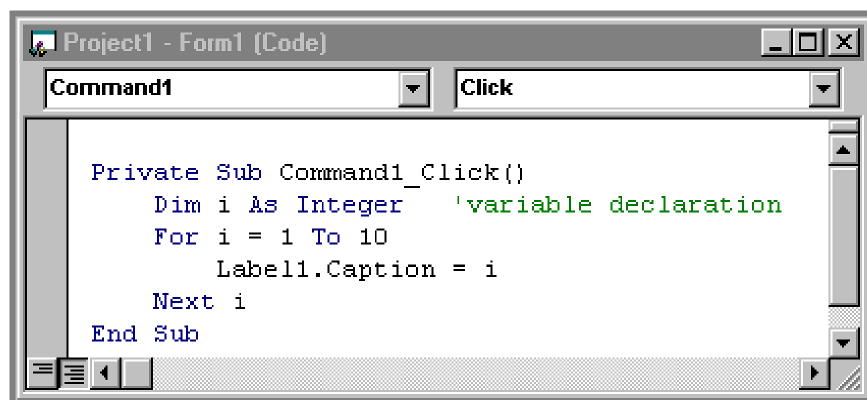
אני ממליץ להקדיש זמן מיוחד ללימוד מערכת העזרה ולהתנסות בדפדוף ובחיפוש מידע בנושאים שונים. זכור, מתכנת טוב אינו בהכרח זוכר את **כל** הפקודות והפונקציות בעל פה, אך הוא בהחלט **יודע** היכן למצוא אותן לכשיצטרך.

הטופס (Form)

במקביל לשני מצבי העבודה שהזכרנו, גם **הטופס** (Form) נמצא בשני מצבים: **מבט ביצוע** (View Object) המדמה מצב זמן ריצה, **ומבט קוד** (View Code) המציג את שורות הקוד.



תרשים 1.2א: טופס במצב ביצוע

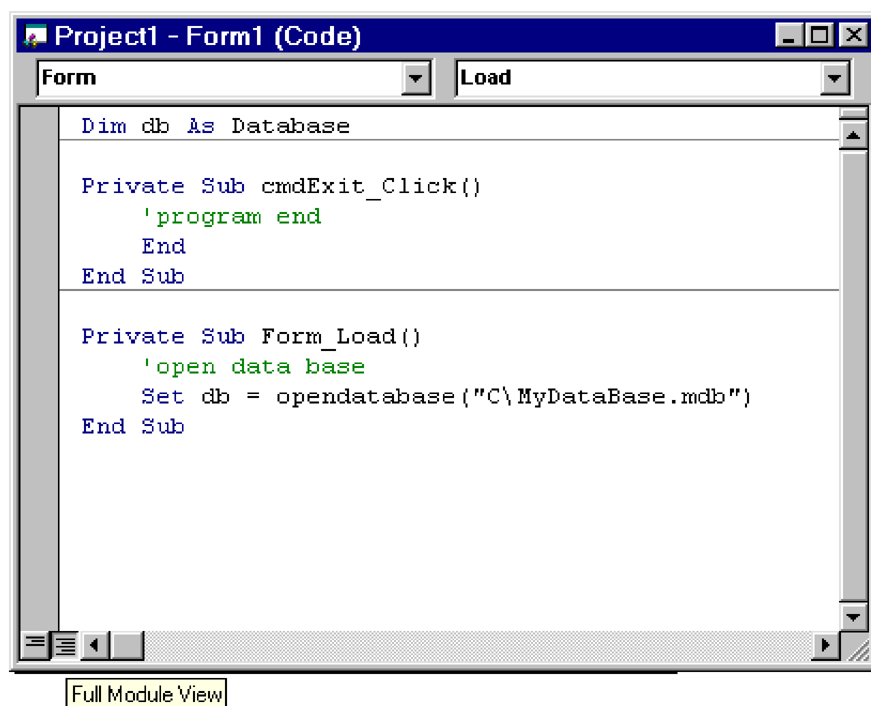


תרשים 1.2ב: טופס במצב קוד

כאשר על הטופס "מפוזרים" פקדים שונים, הוא נמצא במצב עיצוב. כך גם נראה אותו בזמן ריצה וכך יראה אותו המשתמש. "נקודות הרשת" הרבות לא יופיעו, כמובן, בזמן ריצה; הן נמצאות שם בזמן בפיתוח לנוחות המתכנת.

המצב השני, כמודגם בתרשים 1.2ב, הוא מבט קוד. עוד נעסוק במצב זה של הטופס, אך כעת נאמר שיש שתי אפשרויות להתבונן בשורות הקוד.

שורות הקוד בנויות משגרות ופונקציות. בידינו הבחירה לצפות בכל שיגרה בנפרד, או לראות את כל השגרות זו אחר זו כאשר קו מפריד ביניהן. הבחירה מתבצעת באמצעות האפשרות **Options** שבתפריט **Tools**. הכרטיסיה **Editor** פורשת לפנינו אפשרויות שונות. סימון ☑ משמאל לאפשרות **Default to Full Module View** תגרום להצגת כל השגרות כשהן מופרדות זו מזו בקו (תרשים 1.3). ביטול הבחירה יגרום להצגת כל שיגרה בנפרד.

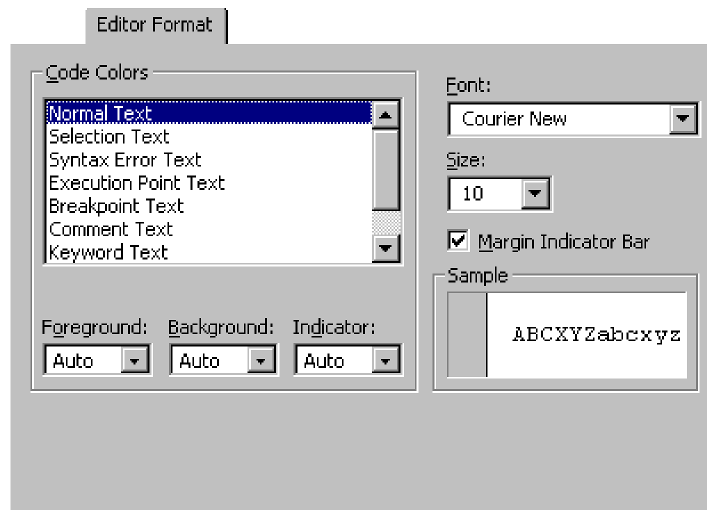


תרשים 1.3: הצגת כל השגרות זו אחר זו כאשר קו מפריד ביניהן

אפשרות נוספת להחלפה מהירה ופשוטה בין שני המצבים האלה נעשית בעזרת שני הלחצנים שבפינה השמאלית התחתונה של הטופס. הלחצן הימני (שבו פסים אחדים) מציג את כל השגרות מופרדות בקו - **Full Module View**, והלחצן השמאלי (בעל שני פסים) מציג כל שיגרה בנפרד - **Procedure View**.

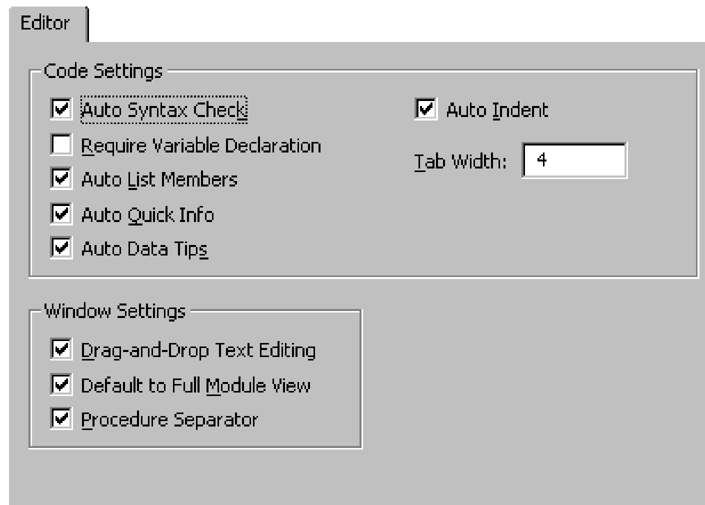
עורך הקוד (Editor)

כשהטופס במבט קוד, הוא למעשה **עורך** (Editor) שבו אנו עורכים את שורות הקוד המרכיבות את התוכנית. באפשרותנו לעצב באופן אישי את מראה העורך כרצוננו ולנוחותנו. העיצוב כולל שינוי גופן התווים, שינוי גודל וצבע. בנוסף, ניתן לשנות את צבע הרקע של משטח העורך, ועוד. את השינויים ניתן לבצע על ידי בחירה בכרטיסיה **Editor Format** באפשרות **...Options** שבתפריט **Tools** (תרשים 1.4).



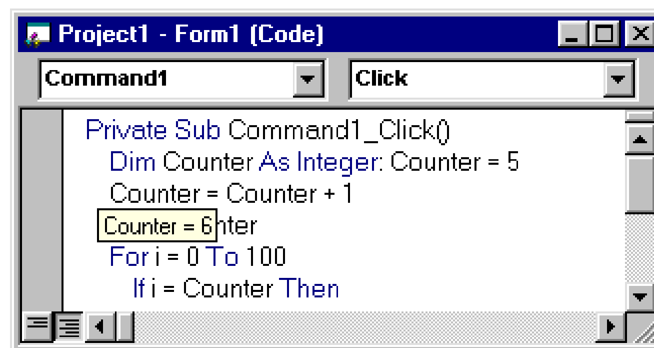
תרשים 1.4

בכרטיסיה **Editor** (תרשים 1.5) נוכל לקבוע ולהגדיר לעורך הגדרות שונות.



תרשים 1.5

- **Auto Syntax Check** - קובע אם ויז'ואל בייסיק תתריע על שגיאות תחביר בקוד שנכתב.
- **Require Variable Declaration** - קובע האם חובה להצהיר על משתנים לפני השימוש בהם בקוד, או לא.
- **Auto ListMembers** - קובע האם תיבת רשימה (ראה תרשים 2.7) תופיע באופן אוטומטי (בזמן שמקישים נקודה לאחר שם הפקד), והאם היא תכלול את כל המאפיינים והשירותים השייכים לאותו פקד, או לא.
- **Auto Quick Info** - קובע האם יופיע חלון מידע על הפרמטרים של הפונקציה (ראה תרשים 2.8) בכל פעם שכותבים את שמה בקוד, או לא.
- **Auto Data Tips** - מציג את ערך המשתנה (בזמן ריצה) בכל זמן שסמן העכבר מוצב על שמו, כדוגמת התרשים הזה:



תרשים 1.6

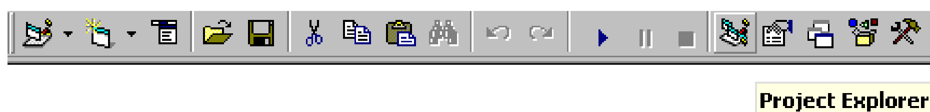
סרגל הכלים (ToolBar)

סרגל הכלים (ראה תרשים 1.7) מכיל לחצני קיצור לפעולות שכיחות. את כל האפשרויות שמספק סרגל הכלים מספקת גם שורת התפריטים (ואפילו יותר), אך פעולות שכיחות החוזרות על עצמן מצויות כסמלים על הסרגל לנוחות המתכנת.



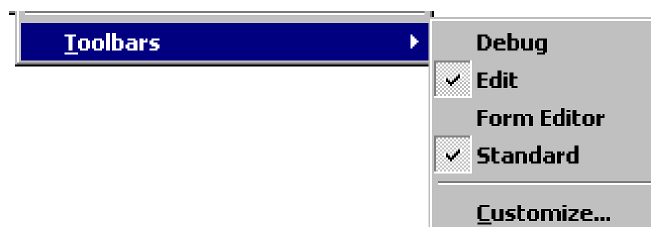
תרשים 1.7: סרגל הכלים

כדי לדעת מהי משמעות כל לחצן בסרגל הכלים, הצב את סמן העכבר על הלחצן (ללא לחיצה עליו). לאחר שנייה או שתיים תתקבל תיבת טקסט (ראה תרשים 1.8) עם שם הלחצן ומשמעותו - וזהו **תיאור כלי** (**ToolTip**).

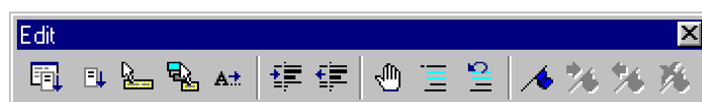


תרשים 1.8

הסרגל המוצג בתרשימים הוא הסרגל הסטנדרטי. ניתן להוסיף לסרגל כלים נוספים, כפי הצורך. בחירה באפשרות **Toolbars** מתפריט **View** מציגה את התפריט הזה:

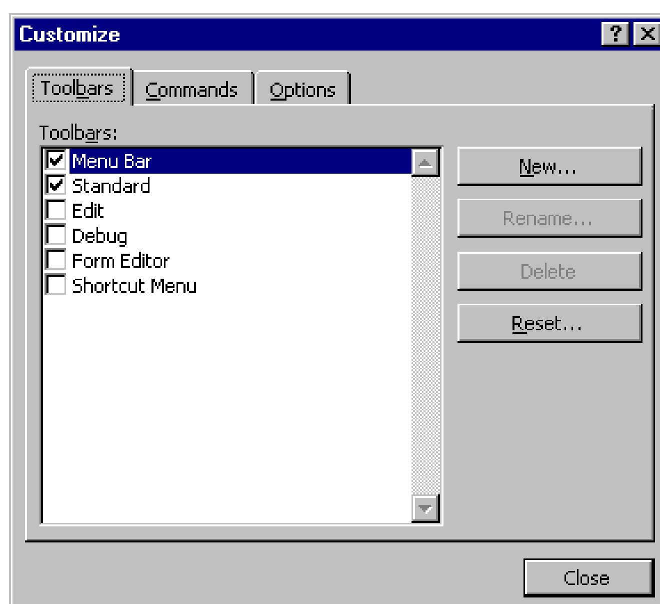


התפריט מאפשר להוסיף ולהסיר סרגלים שונים. בחירה באפשרות **Edit** תגרום להוספת הסרגל הזה:



סרגל זה מסייע בעיקר לעריכת שורות הקוד.

בחירה באפשרות **Customize** תציג את החלון הזה:



מחלון זה ניתן לערוך ולהתאים אישית את סרגל הכלים ולהתאימו לדרישות.

בכרטיסיה **Toolbars** ניתן ליצור, לשנות שם, למחוק או לאתחל מחדש את סרגל הכלים.

הכרטיסיה **Commands** מכילה רשימת פעולות אשר ניתן לגרור אותן אל סרגל הכלים או אל שורת התפריטים, וכך להוסיף אותן באופן ידני.

הכרטיסיה **Option** מאפשרת לשנות את גודל הלחצנים, להוסיף לתיאור הכלי (המופיע עם הנחת סמן העכבר על פני הלחצן) את מקש הקיצור המקביל בפעולתו ללחצן. בנוסף, באמצעות כרטיסיה זו ניתן להוסיף "תנועה" לשורת התפריטים, כך שלחיצה על אחד התפריטים בשורה תגרום להצגת התפריט כולו בתנועה הדרגתית ולא בצורה פתאומית.

את סרגל הכלים ניתן להציג או להסתיר על ידי האפשרות **Toolbars** שבתפריט **View**.

ארגז הכלים (ToolBox)



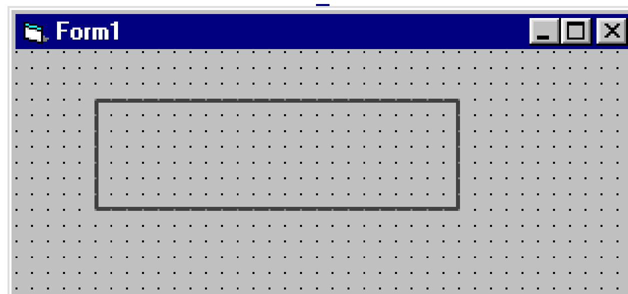
תרשים 1.9: ארגז הכלים

כל דבר הוא "**אובייקט**". "**פקד**" הוא סוג מסוים של אובייקט.

בארגז הכלים מצויים **פקדים** (**controls**) שונים שניתן לשלבם בטופס. כל פקד הוא מחלקה (class) המספקת שירותים, והוא מיוצג לרוב על ידי קובץ **ocx**. לארגז הכלים ניתן להוסיף באופן דינמי פקדים נוספים, ועל כך בפרק 3.

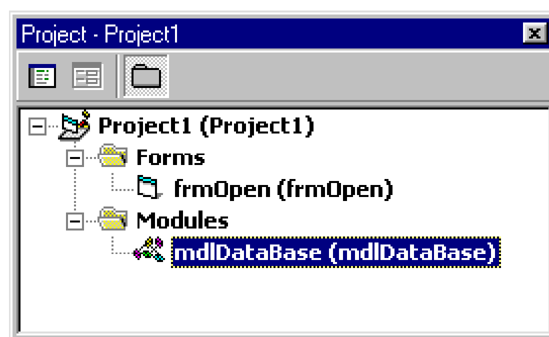
אם ארגז הכלים אינו מופיע על המסך, ניתן להציגו בכל רגע בבחירת האפשרות **Toolbox** שבתפריט **View**.

בפרק 3 נעסוק בהרחבה בפקדים השונים ובתפקידיהם. ככלל, העבודה עם ארגז הכלים מתבצעת בבחירת הפקד מהארגז ותיחום מקומו באזור הרצוי בטופס.



תרשים 1.10: קביעת מקום הפקד על פני הטופס

חלון הפרויקט (Project Explorer)



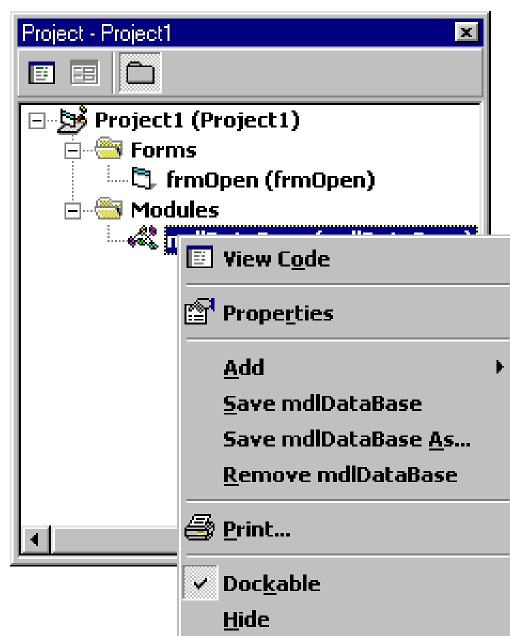
תרשים 1.11: חלון הפרויקט

חלון הפרויקט (Project Explorer), ולמעשה - סייר הפרויקט, מציג את כל הקבצים הכלולים בפרויקט. יישום ויז'ואל בייסיק מורכב לרוב מ**טפסים** - **Forms** (קבצי **frm**) ו**מודולים** - **Modules** שהם קבצי הגדרות (קבצי **bas**). כלומר, כל טופס הוא קובץ המאופיין בסיומת frm והמודול הוא קובץ בעל סיומת bas. את כל הקבצים האלה (frm ו-bas) מאגד קובץ אחד - **קובץ הפרויקט**, המאופיין בסיומת **vbp**. פרויקט בוויז'ואל בייסיק יכול להכיל סוגי קבצים נוספים.

יישום המכיל שני טפסים (Forms) ומודול (Module) אחד, יכלול למעשה ארבעה קבצים שונים: שני קבצי frm לשני הטפסים, קובץ bas אחד למודול, וקובץ vbp אחד שהוא קובץ הפרויקט. מלבד קבצי frm ו- bas יכולים להשתתף בפרויקט קבצים מסוגים נוספים ואף הם יוצגו בחלון זה.

בחלון הפרויקט (Project Explorer) נמצא את שם הפרויקט ואת שם הקובץ בו הוא נשמר (בסיומת vbp). תחתיו מוצגת רשימת הטפסים ושמות הקבצים (frm), ולבסוף שמו של המודול והקובץ בו הוא שמור (bas). בראש החלון, תחת הכותרת המכילה את שם הפרויקט, נבחין בשלושה לחצנים. הימני - בעל צורת תיקיה ב- Windows 95 - מציג את רשימת הקבצים בחלון בפורמט ה"סייר" של Windows 95 (ומכאן שמו). משמאל נמצא הלחצן View Object, אשר לחיצה עליו מביאה את הטופס למבט עיצוב, שבו נוכל לראות את הפקדים השונים המרכיבים את הטופס. הלחצן השמאלי ביותר - View Code - מציג את הטופס במבט קוד.

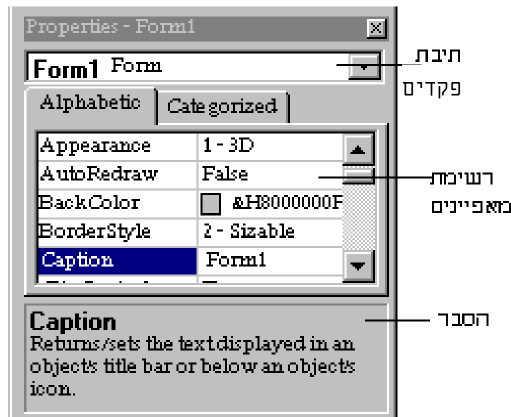
מחלון זה ניתן לשמור את הקבצים, או לחילופין למחוק אותם מהפרויקט. לחיצה ימנית בעכבר כשהוא מצביע על אחד מהקבצים, תציג תפריט אישי עם מיגוון אפשרויות (ראה תרשים 1.12). ביניהן האפשרויות **Save** ו-**Remove** השומרות ומוחקות את הקבצים בהתאמה.



תרשים 1.12

את חלון הפרויקט ניתן להציג בכל רגע בבחירת האפשרות **Project Explorer** שבתפריט **View**, או בלחיצה על המקשים **Ctrl+R**.

חלון המאפיינים
(Properties Window)



איור 1.9: חלון המאפיינים

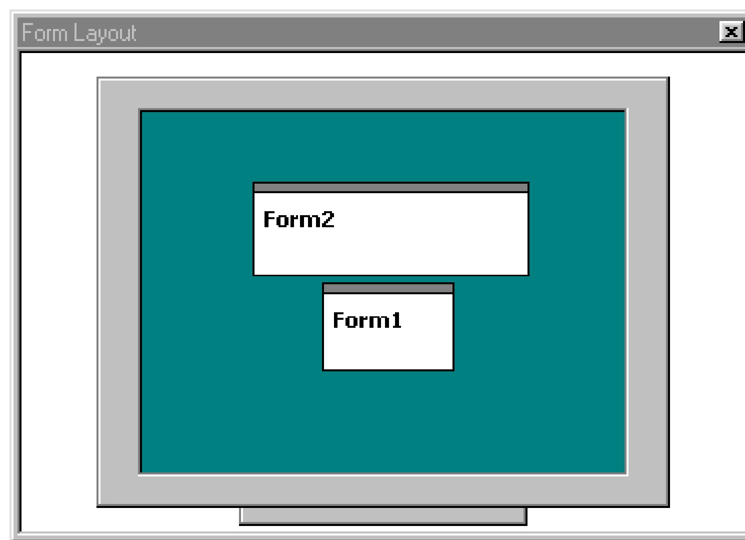
חלון המאפיינים (Properties Window) מכיל את כל מאפייני הפקד. לפקדים שונים יש גם מאפיינים שונים; יש מאפיינים ייחודיים לפקדים שונים, אך בדרך כלל רוב המאפיינים משותפים לכל הפקדים. בראש החלון נמצאת תיבה נפתחת - **תיבת הפקדים**, אשר מכילה את רשימת כל הפקדים שעל פני הטופס. בחירה באחד מהפקדים שבתיבה תשנה בהתאמה את תוכן רשימת המאפיינים, לזו התואמת לפקד הנבחר. כלומר, רשימת המאפיינים משתנה בהתאם לפקד שנבחר מהרשימה.

עדכון ערך המאפיין נעשה באופן פשוט ביותר. גוללים את הרשימה עד להופעת המאפיין המבוקש. המאפיינים מסודרים בכרטיסיה אחת לפי קטגוריות ובכרטיסיה שנייה לפי סדר הא״ב, ומקלים בכך על החיפוש. מצד שמאל יוצג שם המאפיין ומצידו הימני יוצג ערכו (מספר, טקסט, שם קובץ וכדו׳). כל שנשאר לעשות הוא, למחוק את הערך הקודם ולעדכן בערך חדש. במאפיינים מסוימים נתבקש לבחור ערך מטווח ערכים מוגדר מראש. במקרה כזה לא ניתן להקנות ערך חדש השונה מהאפשרויות המוצעות. על משמעות המאפיינים נלמד בפרק 4.

גם את חלון המאפיינים ניתן להציג בכל רגע בבחירת האפשרות **Properties Window** מתפריט **View**, או להקיש **F4**.

חלון המתאר
(Form Layout Window)

חלון המתאר (Form Layout Window) מציג בצורה גרפית מסך קטן המדמה את המסך האמיתי. באמצעות מסך גרפי זה ניתן למקם את כל הטפסים בפרויקט ולבחון את מיקום כל טופס ביחס לטפסים האחרים. באותה צורה בה מוצגים הטפסים בחלון כך הם יוצגו למשתמש. תרשים 1.13 מדגים תצוגה של שני טפסים. במקרה זה Form2 יהיה מעל Form1 באותו מיקום, באותו גודל ובאותו יחס.

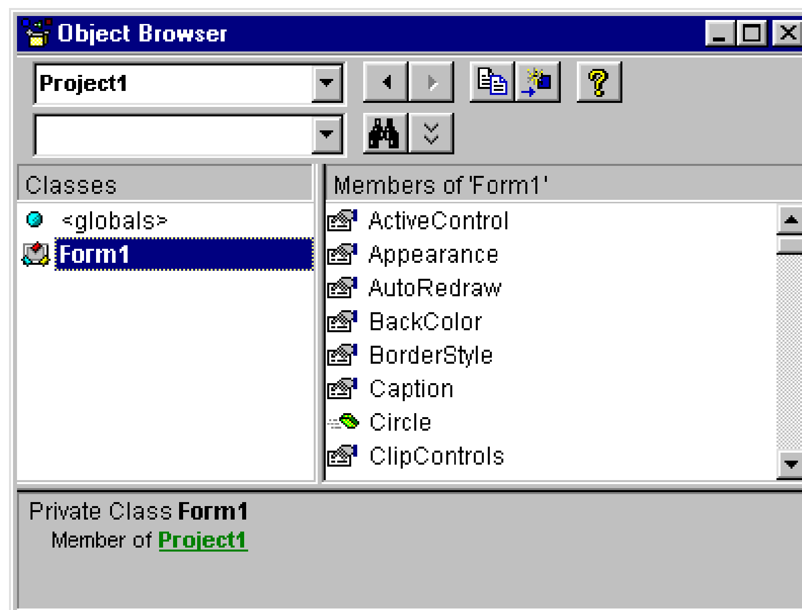


תרשים 1.13: חלון המתאר

להצגת חלון המתאר בחר באפשרות **Form Layout Window** שבתפריט **View**.

דפדפן האובייקטים
(Object Browser)

תפקיד דפדפן האובייקטים לספק מידע על הפקדים הקיימים בוויז'ואל בייסיק וביישומים נוספים. בנוסף, ניתן לראות דרכו את כל השירותים והמאפיינים השייכים לפקדים אלה, לקבל עליהם עזרה ולהדביקם לתוך שורות הקוד ביישום.

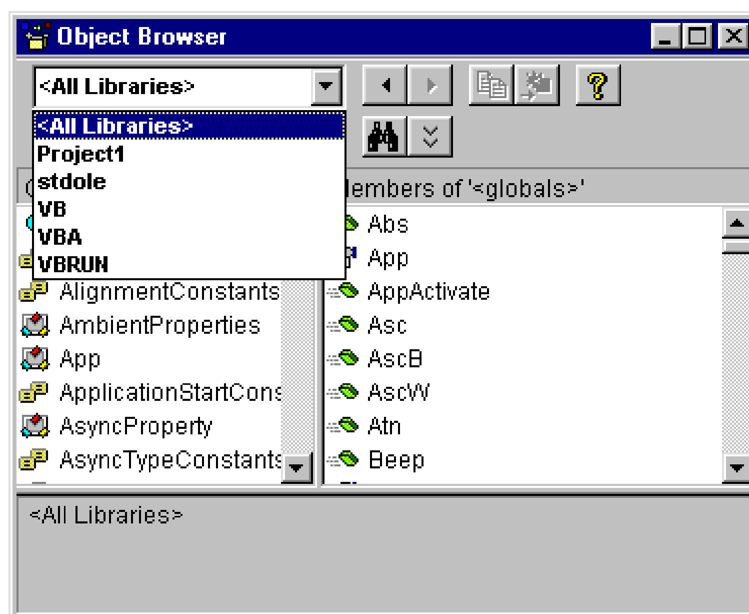


תרשים 1.14: דפדפן האובייקטים

הפריטים בדפדפן האובייקטים מספקים מידע אודות מאפיינים, שירותים, שגרות וקבועים אלה:

1. Projects - פרויקטים המוגדרים על ידי המתכנת.
2. stdole (Standard OLE) - פקד OLE.
3. VB (Visual Basic) - built in בוויז׳ואל בייסיק.
4. VBA (Visual Basic for Application) - שפת ויז׳ואל בייסיק ליישומים.
5. VBRUN (Visual Basic Run Time) - זמן ריצה של ויז׳ואל בייסיק.

רשימה זו מופיעה בתיבה נפתחת שנמצאת בחלק השמאלי העליון של חלון הדפדפן, כמודגם בתרשים 1.15.

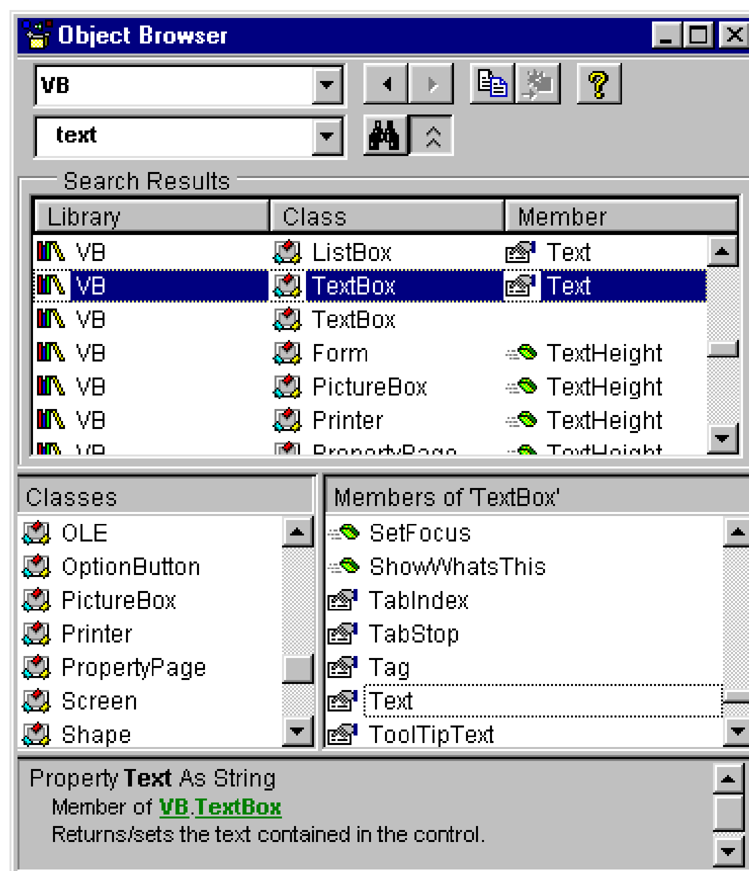


תרשים 1.15

תיבת הטקסט הנפתחת בחלון Project\Library היא **Search Text**. בתיבה זו ניתן להקליד מילה אחת או יותר, ובלחיצה על לחצן **Search** להתחיל בחיפוש. חלון הדפדפן מתחלק לשלושה חלקים (ראה תרשים 1.16).

כדוגמה הקש את המילה "text" ולחץ על לחצן החיפוש (כמודגם בתרשים 1.16). בחלון מופיעים כל המאפיינים והשירותים המכילים את המילה "text". בכל שורה של תוצאת החיפוש (Search Results) מופיע מידע על מקור המאפיין, השירות או הפקד (Library), המחלקה אליה הוא משתייך (Class) וגם המאפיינים והשירותים, כאשר המילה היא מאפיין או שירות (Member).

לחיצה על השורה תגרום להצגת כל המאפיינים והשירותים בחלק האמצעי של חלון הדפדפן (כאשר היא מתייחסת לפקד, לדוגמה). לחיצה על אחד המאפיינים או השירותים האלה תגרום להצגת הסבר קצר בחלקו התחתון של החלון, על המאפיין או השירות שנבחר.



תרשים 1.16

באמצעות הלחצן **Copy To Clipboard** ניתן להעתיק את שם הפקד או את שם המאפיין או השירות שנבחר, ולהדביקו בשורות הקוד של ויז'ואל בייסיק.

בחירה באחד המאפיינים או השירותים ולחיצה על לחצן **Help**, תגרום להצגת חלון עזרה בנושא הנבחר.

כדי להציג את דפדפן האובייקטים יש לבחור באפשרות **Object Browser** שבתפריט **View**, או להקיש **F2**.

2:
צעד ראשון בפיתוח יישום

עד כה עסקנו בהיבט התיאורטי של הכלים העומדים לרשותנו. בפרק זה נצעד צעד מעשי ראשון לפיתוח היישום הראשון שלנו בוויז'ואל בייסיק. נבחן את תהליך פיתוח היישום מתחילתו ועד סופו. בשלב זה של הלימוד לא ניצור קובץ הרצה וערכת התקנה; נשאיר זאת לפרק המסיים את ספרנו.

בראש ובראשונה יש לדעת איזה יישום אנו עומדים לפתח, מה ביכולתו לבצע ומה מצפה ממנו המשתמש. כלומר, מה המשתמש רוצה להשיג. נתחיל את העבודה בתכנון מקדים על הנייר ולאחר מכן ניישם את אשר למדנו בפרק המבוא לספר, שפיתוח כל יישום בוויז'ואל בייסיק צריך לכלול ארבעה שלבים:

א. יצירת ממשק משתמש.

ב. קביעת מאפיינים.

ג. כתיבת הקוד.

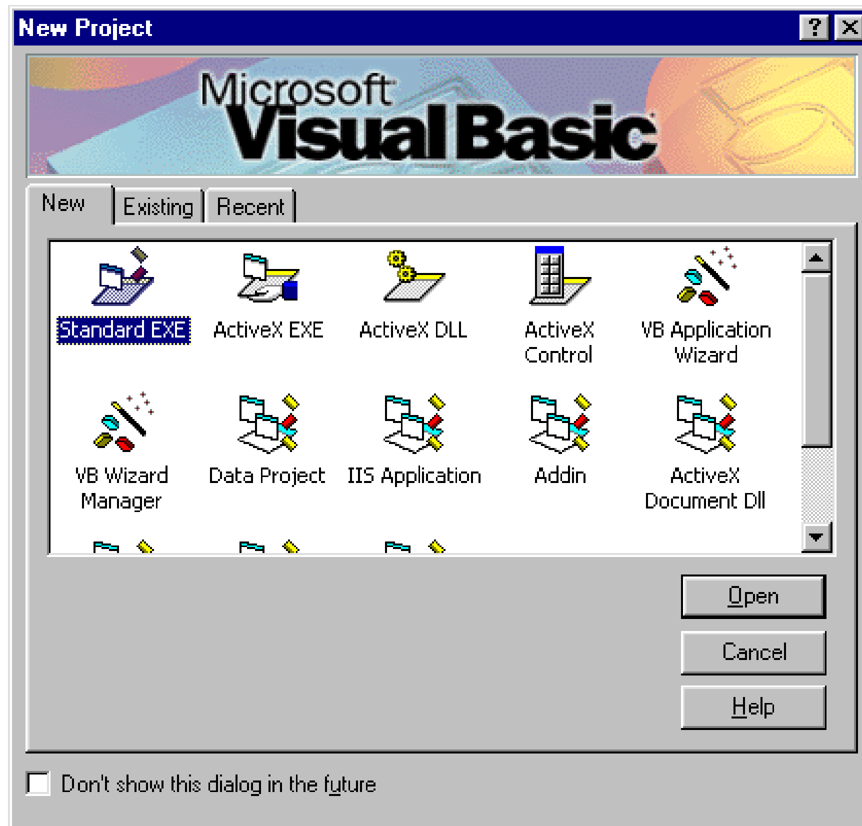
ד. יצירת קובץ הרצה (וערכת התקנה).

נחליט כי היישום הראשון שלנו יהיה הדוגמה שהצגנו בפרק המבוא, בו נאפשר למשתמש להכניס מספר כלשהו. כאשר הוא ילחץ על הלחצן **העלאה בחזקה** תוצג לפניו החזקה הריבועית של המספר שבחר.

עתה נפנה למחשב ונפעיל את תוכנת ויז'ואל בייסיק. הרצת ויז'ואל בייסיק תיעשה בסדר הבא:

1. לחץ על לחצן **התחל** Start (Start בגירסה האנגלית).
2. בחר באפשרות **תוכניות** (Programs בגירסה האנגלית).
3. בחר באפשרות **Microsoft Visual Basic**.
4. בחר באפשרות **ויז'ואל בייסיק**.

אחרי שהתוכנה סיימה את הטעינה לזיכרון, מופיע חלון דו-שיח **New Project** כמודגם בתרשים 2.1:



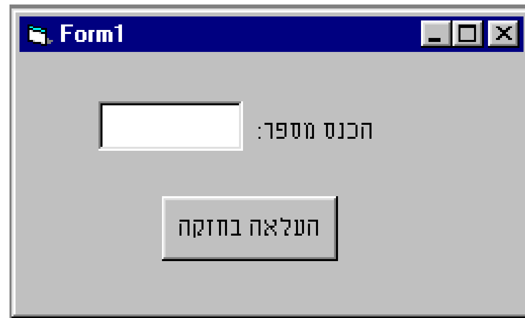
תרשים 2.1

בחלון זה יש שלוש כרטיסיות: **New** - לפתיחת פרויקט חדש, **Existing** - לפתיחת פרויקט קיים, **Recent** - לפתיחת פרויקט שרץ לאחרונה.

מכיון שזהו הפרויקט הראשון שלנו, נבחר בכרטיסיה **New**, ובה נבחר בסוג קובץ **Standard EXE**. לחיצה על לחצן **Open** תפתח פרויקט חדש ותכניס אותנו לסביבת הפיתוח של ויזואל בייסיק.

יצירת ממשק המשתמש

בסיום בניית ממשק המשתמש, הטופס אמור להיראות כמודגם בתרשים 2.2.



תרשים 2.2

כדי ליצור את ממשק המשתמש נבצע את השלבים הבאים:

1. נבחר מארגז הכלים את האובייקט **Label**.
2. נציב את האובייקט במקומו על הטופס (ראה תרשים 2.2). אם שטח הפנים של האובייקט גדול או קטן מדי, ניתן לשנות את גודלו בגרירת אחת משמונה הנקודות הסובבות את האובייקט (כפי שאנו נוהגים במקרים אחרים). אם נקודות אלו אינן קיימות, לחיצת עכבר בשטח האובייקט תגרום להופעתן.
3. באופן דומה נבחר את **TextBox** ונמקם אותו לשמאל התווית - Label.
4. כך נפעל גם עם הלחצן, הוא הפקד **CommandButton**. ניתן להזיז אובייקט ממקום למקום בטופס על ידי גרירתו.

בהצבת שלושת האובייקטים במקומם, סיימנו את שלב יצירת ממשק המשתמש ונעבור עתה לשלב הבא, קביעת המאפיינים.

קביעת המאפיינים

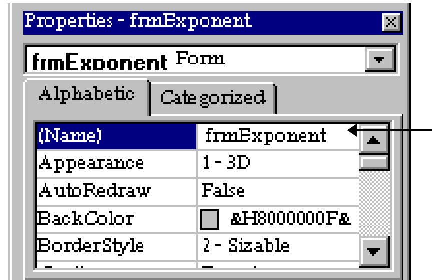
תוכל לקבוע ערכים למאפייני האובייקטים, בשתי דרכים: באמצעות חלון המאפיינים או בשורות קוד. בספר זה נסגל לעצמנו את הכלל הבא: כאשר אנו קובעים ערכים ראשוניים למאפיינים, נקבע אותם באמצעות חלון המאפיינים. כדי לשנות ערכי מאפיינים באופן דינמי נשתמש בשורות קוד.

יש לוודא שחלון המאפיינים מופיע על המסך. פתיחת תיבת האובייקטים הנגללת תגלה לנו 4 אובייקטים. שלושה מהם הצבנו בעצמנו על הטופס, והרביעי הוא הטופס (Form) עצמו.

נתחיל בתהליך קביעת המאפיינים בסדר הבא:

מתוך תיבת הפקדים נבחר ב- Form1.

נשנה את ערך המאפיין Name מ-Form1 ל-frmExponent כמודגם בתרשים 2.3.



תרשים 2.3

המאפיין Name קובע את שם הפקד. שם זה נציין בכל פעם שנבצע פעולה הקשורה בפקד. סיבת שינוי התכונה Name מ-Form1 ל-frmExponent נובעת ממניעים מקצועיים. בתכנות מקצועי שמות הפקדים מקבלים משמעות המבטאת את תפקידם, ובפרט כאשר יש פקד אחד החוזר על עצמו מספר פעמים בטופס. נניח, לדוגמה, כי לצורך קליטת פרטי סטודנט במחשב האוניברסיטה נשתמש בשלושה פקדים מסוג TextBox. במקום ליצור שלושה פקדים Text1, Text2 ו-Text3 לקליטת שם הסטודנט, כתובתו ומספר הטלפון שלו, ניטיב לעשות אם נקבע לשלושת הפקדים את השמות txtName, txtAddress ו-txtTelephone, בהתאמה. הנוחות והמקצועיות יבואו לידי ביטוי בעיקר בפיתוח הקוד, בהן נעשה שימוש רב בשם הפקד.

צירוף הקידומת txt לשם הפקד מסוג TextBox וקידומת frm לשם הטופס, הוא מוסכמה בין מתכנתי ויז'ואל בייסיק, וכך ננהג גם אנו.

בטבלה 2.1 מרוכזת רשימת הקידומות המקובלות לחלק מהפקדים.

טבלה 2.1

פקד	משמעות	קידומת
CheckBox	תיבת סימון	chk
ComboBox	תיבת משולבת	cbo
CommandButton	לחצן פקודה	cmd
CommonDialog	תיבת דו-שיח	dlg
Data	נתונים	dat
DBCombo	תיבת משולבת מוצמדת	dbc
DBGrid	רשת מוצמדת	dbg
DBList	תיבת רשימה מוצמדת	dbl

פקד	משמעות	קידומת
DirListBox	תיבת רשימת קבצים	fil
DirListBox	תיבת רשימת ספריות	dir
DriveListBox	תיבת רשימת כוננים	drv
Frame	מסגרת	fra
HscrollBar	פס גלילה אופקי	hsb
Image	דמות	img
Label	תווית	lbl
Line	קו	lin
ListBox	תיבת רשימה	lst
Menu	תפריט	mnu
OLE	אובייקט מקושר	ole
OptionButton	לחצן אפשרות	opt
PictureBox	תיבת תמונה	pic
Shape	צורה	shp
TextBox	תיבת טקסט	txt
Timer	שעון עצר	tmr
VScrollBar	פס גלילה אנכי	vsb

כפי ששינינו את המאפיין Name, נקבע בטופס (frmExponent) למאפיין StartUpPosition את הערך השני המוצע CenterScreen - 2. מאפיין זה קובע את מיקום הטופס במסך. הערך שבחרנו יגרום להצבת הטופס במרכז המסך. באופן דומה נמשיך עם שאר הפקדים בטופס, בהתאם לטבלה 2.2.

טבלה 2.2

פקד	מאפיין	ערך
Text1	Name Text	txtNum (empty)
Label1	Name Caption	lblNum הכנס מספר:
Command1	Name Caption	cmdExponent העלאה בחזקה

שים לב, כי הערך "Text1" הכתוב במאפיין Text של האובייקט txtNum נמחק ונשאר ריק. הסיבה לכך היא, שאנו רוצים שבשעה שהיישום יפעל התיבה txtNum תהיה ריקה ולא תופיע בה המילה "Text1", וכך שהמשתמש יוכל להקליד בה מספר כרצונו. כמו כן, המאפיין Name של הפקד Label1 שונה ל- lblNum, למרות שניתן לוותר על כך. בעתיד לא נצטרך לפנות אל פקד זה משורות הקוד, כי זהו ערך קבוע, אולם מסיבה מקצועית נשנה גם אותו.

נרחיב מעט את הדיבור על טבלה 2.2. המאפיין Text קובע מהו הטקסט אשר יהיה כתוב בתוך תיבת הטקסט (TextBox). במקרה שלנו, כפי שכבר הוזכר, פקד txtNum משמש לקליטת מספר ולא להודעה, ולכן השארנו תיבה זו ריקה. המאפיין Caption קובע את כותרת הפקד Label (מה שיהיה רשום בו). מאפיין זו מקביל למאפיין Text באובייקט מסוג TextBox. בפקדים Label ו-CommandButton לא קיים המאפיין Text, אלא רק המאפיין המקביל לו - Caption. לכך, בדוגמה שלנו, נקבע המאפיין Caption של הפקדים lblNum ו-cmdExponent ל-"הכנס מספר:" ו-"העלאה בחזקה", בהתאמה.

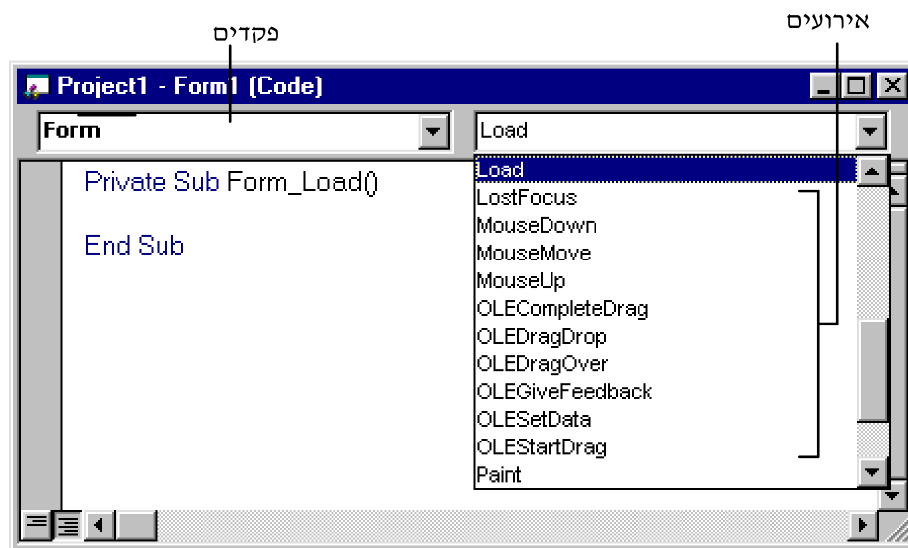
לשאר המאפיינים לא התייחסנו, כי אין צורך בכך. לצורך יישום הדוגמה שלנו מספיק לשנות רק את המאפיינים שהתייחסנו אליהם.

השלב השלישי הינו כתיבת הקוד.

כתיבת הקוד

פרויקט בוויז'ואל בייסיק אינו מחייב בהכרח כתיבת שורות קוד. יישומים פשוטים יכולים לפעול בהצלחה ולשביעות רצון גם ללא שורת קוד אחת. אולם, היישומים המקצועיים אליהם אנו שואפים, יחייבו בוודאי כתיבת שורות קוד לא מעטות. גם ביישום שלנו, על אף פשטותו, עלינו לכתוב שורות קוד. בסעיף זה נדון במושג **מבט קוד** (View Code) של הטופס.

לכתיבת שורות הקוד עלינו להיעזר במרכיבי שפת ויז'ואל בייסיק (או במילים אחרות בתחביר - Syntax של השפה), אשר נלמד במהלך הפרקים הבאים (פרקים 5 עד 11). אולם את עקרונות השימוש בחלון הקוד נלמד כאן ועכשיו. לחיצה על לחצן View Code (השמאלי ביותר) שבחלון הפרויקט (Project Explorer) מביאה אותנו למבט קוד של הטופס, כמודגם בתרשים 2.4.



תרשים 2.4: מבט קוד - View Code

בחלק העליון של חלון הקוד נמצאות שתי תיבות משולבות (נפתחות) Objects ו-Procedure. התיבה השמאלית Objects מכילה את רשימת כל הפקדים הקיימים בטופס שיצרנו, כולל הטופס עצמו. בראש הרשימה מופיעה המילה (General). מילה זו אינה מציינת פקד, והיא מפנה אותנו להגדרות הכלליות של הטופס (להגדרות אלו נתייחס מאוחר יותר). אם אין לנו הגדרות כלליות, לא תהיה למילה זו משמעות עבורנו.

התיבה הימנית Procedure (בתרשים 2.4 היא פתוחה), מכילה את רשימת האירועים להם יודע להגיב כל פקד ופקד.

תכנות מוכוון אירועים

נעשה אתנחתא קצרה ונסביר את המושג **אירוע** (Event). ויז׳ואל בייסיק הינה שפה **מוּנַעַת אירועים** (Event driven). כל פקד יכול להימצא באירועים שונים, או במילים אחרות, יודע להגיב לאירועים מסוימים שהופעלו עליו. ניקח כדוגמה את הפקד CommandButton (לחצן פקודה), אשר ניתן ללחוץ עליו לחיצה אחת עם העכבר. בפעולה זו הפעלנו על הלחצן את האירוע Click. ניתן לשנות את תוכן הפקד TextBox (תיבת טקסט), או במילים אחרות, להפעיל עליו את האירוע Change. בפנותי לפקד זה הפעלתי עליו את האירוע GotFocus, אך ברגע ש״הסרתי את הזרקורים״ ממנו והפניתי אותם לפקד אחר, התרחש לגביו האירוע LostFocus.

בהפעלת האירוע עדיין לא תם העניין. יש להגדיר איזו פעולה תתבצע בשעה שיתרחש האירוע. לחצן פקודה יודע להגיב לאירוע Click, אך מה שיתרחש כתוצאה מלחיצה

עליו – זאת עלינו המתכנתים להגדיר. אין הכרח להתייחס לכל האירועים הקיימים לפקד. אם לא נגדיר לאירוע דבר, הפקד לא יגיב כאשר יתרחש אותו אירוע.

בשלב זה נכנסות שורות הקוד. כשנרצה שפקד יגיב לאירוע מסוים, עלינו לכתוב עבורו את שורות הקוד המגדירות לו, או לתוכנית עצמה, כיצד להגיב ולהתנהג בזמן שהאירוע מתרחש.

ראוי לשים לב לכך, שישנם אירועים המפעילים אירועים אחרים בזמן התרחשותם. לדוגמה, כאשר נוצר האירוע DblClick (לחיצה כפולה), הוא גורם גם להפעלת האירועים MouseDown, MouseUp ו-Click. על כן, כשכותבים קוד מתאים לכל אחד מהאירועים הללו, יש לקחת בחשבון שהקוד של **כל** האירועים ייצא אל הפועל.

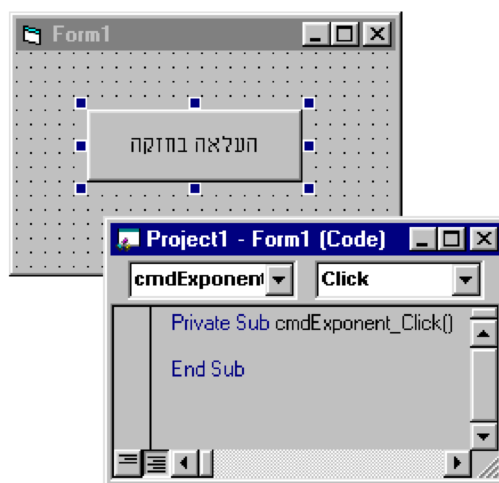
חזרה לכתיבת הקוד

נחזור לתיבת האירועים Procedure. בתיבה זו רשומים כל האירועים שהפקד יודע להגיב להם. התגובה עצמה תלויה בנו ובשורות הקוד שנכתוב לאותו אירוע. ביישום שלנו יש לחצן פקודה **cmdExponent**, אשר בלחיצה עליו אמור המשתמש לקבל כתוצאה את ריבוע המספר שבחר בו. מזכיר משהו? נכון! בפעולת לחיצה זו אנו מפעילים למעשה את האירוע Click על הפקד.

ניישם עתה את מה שלמדנו. בתיבת Object נבחר בפקד cmdExponent ובתיבת Procedure נבחר באירוע **Click**. באופן אוטומטי נקבל את שורות הקוד הבאות:

```
Private Sub cmdExponent_Click()
End Sub
```

לחיצה כפולה על הפקד במבט עיצוב תפתח באופן אוטומטי את חלון הקוד בשיגרה Click של הלחצן (כמודגם בתרשים 2.5).



תרשים 2.5

השורה הראשונה היא שורת ההצהרה על שיגרת האירוע Click, המופעל על הפקד cmdExponent. השורה השנייה End Sub מסיימת את השיגרה. מה שנותר לנו לעשות הוא לכתוב בין שתי שורות אלו את שורות הקוד המתאימות, אשר יורו לתוכנית כיצד לפעול בעת התרחשות האירוע Click.

לפני שניגש למלאכת הכתיבה, ניזכר במה שלמדנו במבוא בדבר קביעת מאפיינים באמצעות קוד. שינוי או קביעת ערך למאפיין יתבצע באופן הבא:

```
lblName.Caption = ”Danny”
```

או עם הפקד TextBox:

```
txtSentence.Text = ”I am learning Visual Basic.”
```

באותו אופן נפעל גם ביישום שלנו. אנו מצפים, שכאשר נלחץ על הלחצן "העלאה בחזקה" תופיע בתיבת הטקסט txtNum התוצאה שהיא ריבוע המספר שהמשתמש הקליד. יוצא, שהמאפיין העומד להשתנות הוא המאפיין Text של האובייקט txtNum. בלחיצה על הלחצן "העלאה בחזקה" מאפיין זה אמור לקבל ערך חדש. לפי זה, תחילת שורת הקוד תיראה כך:

```
txtNum.Text =
```

מהו הערך החדש שיתקבל?

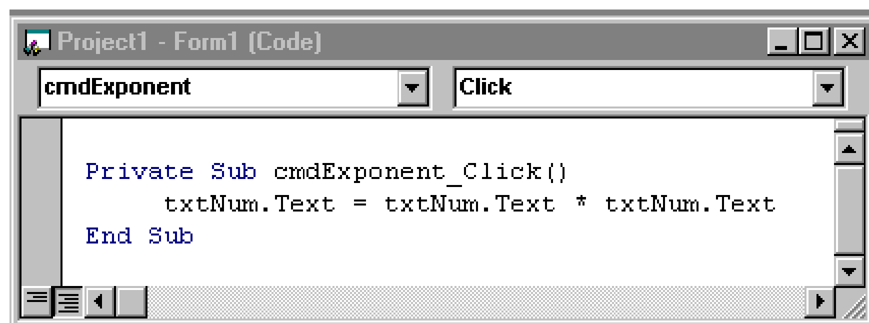
הערך החדש יקבל את התוצאה שהיא הריבוע של הערך הישן, או במילים אחרות הערך הישן כפול עצמו:

```
txtNum.Text * txtNum.Text
```

ובחיבור שני חלקי המשוואה נקבל:

```
txtNum.Text = txtNum.Text * txtNum.Text
```

המחשב מחשב תחילה את האגף הימני במשוואה (עם הערך הישן של txtNum) ורק אחר כך הוא מבצע את פעולת ההשמה באגף השמאלי (ומתקבל הערך החדש). זוהי גם התמונה השלמה כמודגם בתרשים 2.6 (בדוק בהתאמה את חלון הקוד שלך).



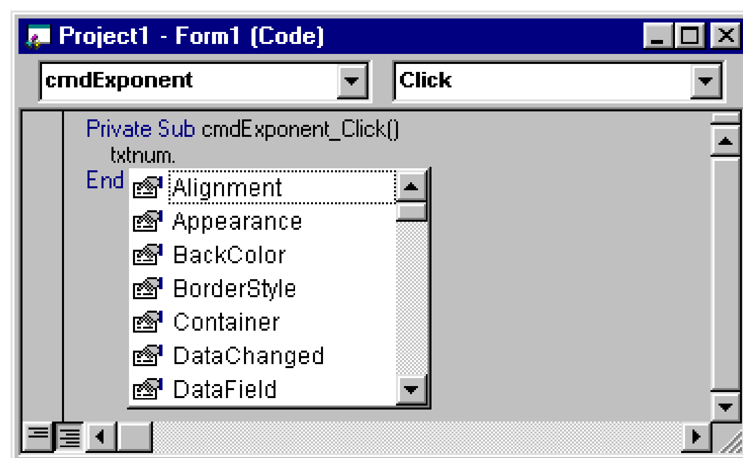
תרשים 2.6

לשמירה על תבנית התכנות נקפיד ללחוץ על מקש Tab ולכתוב את שורות הקוד הפנימיות מעט ימינה (ראה תרשים 2.6).

כל שנותר, הוא להפעיל את היישום (בזמן ריצה של התוכנה). להפעלת היישום לחץ או הקש **F5**. להפסקת פעולת היישום ולחזרה למצב פיתוח לחץ (אם שאלת את עצמך כיצד המשתמש יוכל לצאת מהתוכנית, היעזר בסבלנות והמתן לתרגיל מס' 1 בסוף הפרק).

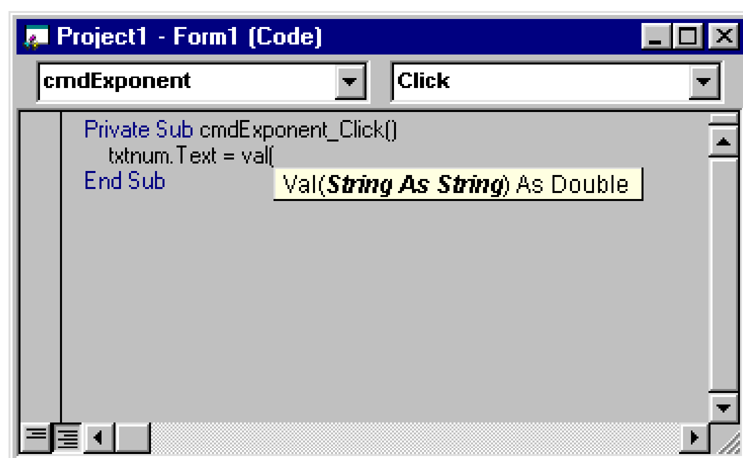
קוד אוטומטי

בוויז'ואל בייסיק קיים קוד אוטומטי, במטרה להקל על כתיבת הקוד. כדי לקבוע כותרת לפקד תווית (Label) יש לקבוע ערך למאפיין Caption. בשורת קוד הדבר יתבטא בכתיבת שם הפקד, נקודה מפרידה ולאחריה שם המאפיין. אך אל דאגה, אין צורך ללמוד את כל רשימת המאפיינים בעל פה. ברגע שכתבנו את הנקודה הבאה לאחר שם הפקד, מופיעה באופן אוטומטי רשימה נפתחת (ראה תרשים 2.7) המכילה את כל המאפיינים (והשירותים) של הפקד. כל שנותר הוא לבחור את המאפיין המתאים וללחוץ על Tab כדי לגרום לכתיבתו בשורת הקוד.



תרשים 2.7

באופן דומה, מתבצעת הפעולה האוטומטית הזו כאשר מתחילים בכתיבת פונקציה כלשהי. ברגע שמסיימים את כתיבת שם הפונקציה, מופיע אוטומטית חלון (ראה תרשים 2.8) המכיל את רשימת הארגומנטים שצריכים להישלח אל הפונקציה.



תרשים 2.8

סדר במבנה הקוד

בכתיבת הקוד חשוב להקפיד על כתיבה ברורה וקריאה. כמתכנתים, פעמים רבות עלינו לפענח קוד של מתכנת אחר, או אפילו קוד שאנו כתבנו לפני זמן רב. אם שורות הקוד לא תהינה קריאות וברורות, קשה יהיה להבין את אשר נכתב כבר, ובוודאי יהיה קשה לשלב קוד חדש בתוכנית.

אחד הדברים החשובים בכתיבת הקוד הוא שימוש בשמות משמעותיים עבור הפקדים, משתנים, שגרות, פונקציות ועוד. כאשר בין שורות הקוד נמצאת השורה הבאה:

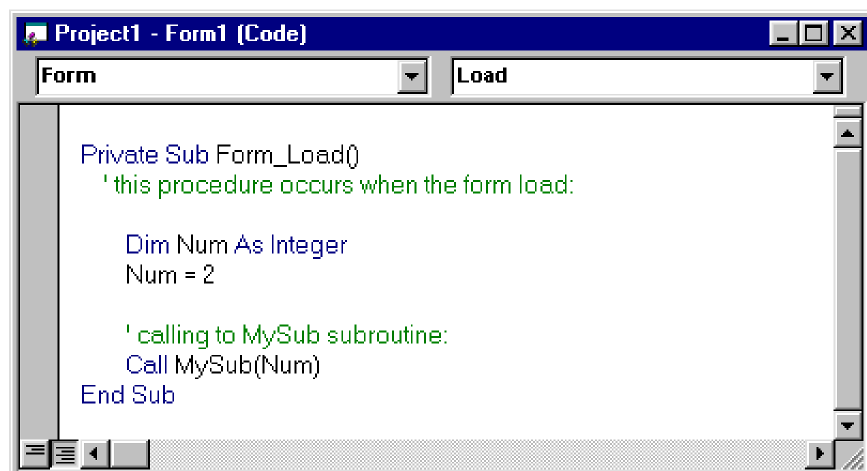
```
txtName.Text = strName
```

נבין כי הפקד `txtName` (מסוג `TextBox`) מציג שם פרטי כלשהו והמשתנה `strName` מכיל אותו. לשורה הבאה, לעומת זאת, אין משמעות רבה עבורנו:

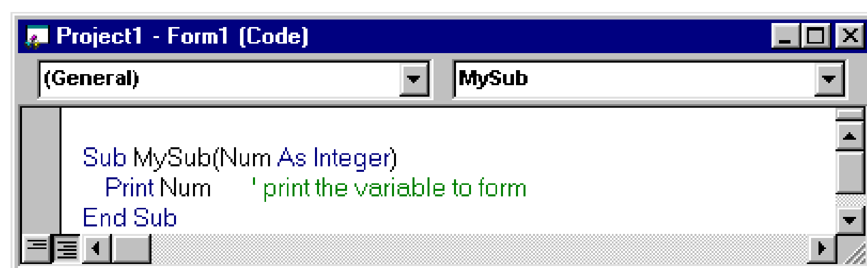
```
Text1.Text = A
```

דבר חשוב נוסף הוא התיעוד של שורות הקוד שאנו כותבים. לפני כל שיגרה מורכבת או שורת קוד שאינה ברורה מייד עם קריאתה, מומלץ להוסיף שורת הערה המסבירה את משמעות שורות הקוד הבאות אחריה. כדי לגרום לשורת קוד להיות מובנת על ידי ויז'ואל בייסיק כ"הערה", נוסיף בתחילתה את התו " ' " (גרש). שורת ההערה נצבעת באופן אוטומטי בירוק (או בכל צבע אחר שהוגדר על ידך), וכאשר ויז'ואל בייסיק מהדרת את היישום אין היא מתייחסת לשורות הערה אלו; היא מדפיסה אותן בלבד.

את ההערה ניתן להוסיף כשורה נפרדת (תרשים 2.9), או בסוף שורת קוד (תרשים 2.10). אין אפשרות "לשתול" הערה באמצע שורת קוד.

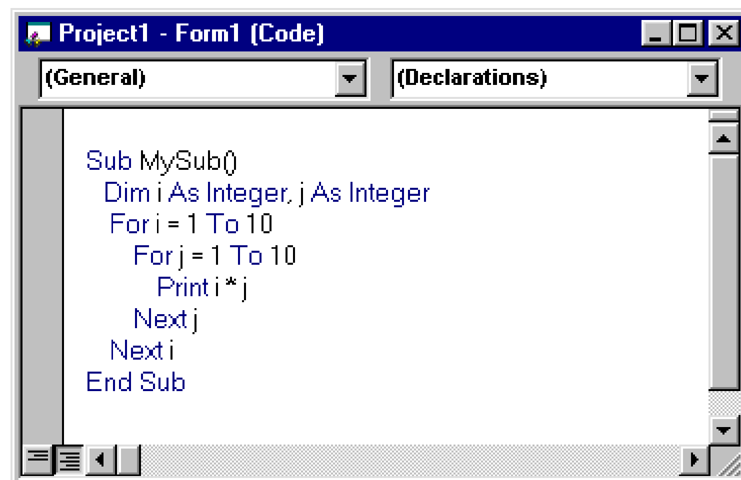


תרשים 2.9



תרשים 2.10

הדבר האחרון ברשימת הדברים שעלינו לעשות הוא השמירה על סדר הכתיבה של שורות הקוד. כתיבה מסודרת כוללת הזחה קלה ימינה (בעזרת מקש Tab) של שורות פנימיות. בנוסף, אם ישנם מבנים מקוננים, יש ל״דחוף״ ימינה כל מבנה בפני עצמו (ראה תרשים 2.11).



תרשים 2.11

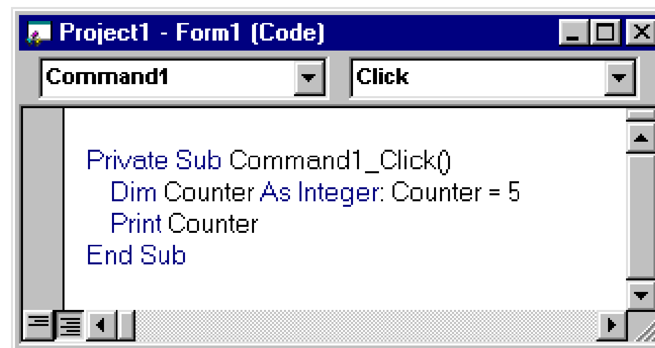
אם צריך לכתוב שורת קוד אחת המכילה פרטים רבים, אין צורך לכתוב את הכל בשורה אחת; ניתן לחלק את השורה על פני מספר שורות בעזרת התו "_" (קו תחתון) בסוף הקטע ה"שבור".

ויז'ואל בייסיק תתייחס לכל השורות האלו כאל שורה אחת, והקוד יהיה קריא יותר (ראה תרשים 2.12).



תרשים 2.12

לחילופין, ניתן לכתוב בשורת קוד אחת שני משפטים שונים (תרשים 2.13). כדי שוויז'ואל בייסיק תדע שלפניה שני משפטים שונים, יש להפריד ביניהם באמצעות נקודתיים (":").



תרשים 2.13

סיכום הנושאים העיקריים שנלמדו בפרק

סקרנו ותרגלנו את ארבעת השלבים בפיתוח כל יישום:

א. יצירת ממשק משתמש.

ב. קביעת מאפיינים.

ג. כתיבת קוד.

ד. הרצת הפרויקט.

הכרנו את האובייקטים Label, TextBox ו- CommandButton.

למדנו כי המאפיין Name קובע את שם הפקד, ושם זה מציינים בכל פנייה אל הפקד. הקפדנו על מתן שמות משמעותיים לפקדים, הכוללים קידומת המעידה על סוג הפקד.

המאפיינים Caption ו- Text של הפקדים Label ו-TextBox בהתאמה, קובעים את התוכן הנכתב בהם.

למדנו מהו **תכנות מוכוון אירועים**. לכל פקד יש אירועים שהוא יודע להגיב אליהם. תפקידנו לכתוב את שורות הקוד אשר יבוצעו כתגובה להתרחשות האירוע.

הפנייה למאפיין של פקד בשורת קוד נעשית על פי הכלל הבא:

ערך = מאפיין.פקד

תרגול

מומלץ מאוד לבצע את כל התרגילים. הם מקנים למתרגל הכשרה וניסיון וגם יכולת חשיבה תכנותית, ובנוסף הם משלימים נושאים שנלמדו בגוף הפרק.

1. שפר את היישום שבנינו יחד במהלך הפרק, והוסף לו לחצן פקודה (CommandButton) שכותרתו "יציאה". לחצן זה, כשמו כן הוא, יאפשר יציאה מהתוכנית.

   הנחיה: הפקודה המסיימת את התוכנית היא End.

2. צור טופס המכיל שלושה לחצני פקודה ותיבת טקסט (TextBox) אחת. לחצן ראשון "Show" יציג את המשפט "Hello World". לחצן שני "Hide" ינקה את תיבת הטקסט והלחצן השלישי "Exit" יסיים את התוכנית.

3. צור את הטופס המוצג בתרשים 2.14.

תרשים 2.14

המשתמש יקליד שני מספרים לשתי תיבות הטקסט השונות. לחיצה על לחצן **Check** תגרום לכך שבתיבה **Result** תוצג תוצאת המכפלה של שני המספרים.

3: פקדים (Controls)

השלב הראשון בכל יישום הוא יצירת ממשק משתמש, ולכן חובה עלינו להכיר היטב את הכלים מהם הוא מורכב. כלי העבודה הם הפקדים (Controls), אשר נמצא אותם **בארגז הכלים** (ToolBox). בפרק זה נלמד להכיר את הפקדים הסטנדרטיים שבארגז הכלים, את תפקידם ואת אופן השימוש בהם. במסגרת זו נכיר גם חלק מהמאפיינים המיוחדים לכל פקד. את שאר המאפיינים, המשותפים לכלל הפקדים, נלמד בפרק הבא.

הכרת הפקדים

ה**פקדים** (Controls) אשר נמצאים בארגז הכלים (ToolBox) הם הכלים המשרתים אותנו ביצירת ממשק המשתמש.

	Pointer	המצביע (Pointer) אינו פקד. תפקידו להביא את העכבר למצבו הרגיל המאפשר תנועה במסך, שינוי גודל חלונות ופקדים שעל הטופס.
	PictureBox	פקד זה מציג תמונות מסוג bitmaps, icons ו-windows metafiles ואחרים.
	Label	תפקיד הפקד להציג טקסט (בדרך כלל כותרת, או הוראה למשתמש). המשתמש אינו יכול לשנות את הטקסט הכתוב בפקד.
	TextBox	פקד זה מקצה מסגרת לכתיבת טקסט, או לשם הצגתו. בניגוד לקודמו, המשתמש יכול לערוך את הטקסט שבפקד.
	Frame	פקד זה מכיל בתוכו באופן ויזואלי ופונקציונלי פקדים אחרים. לכל הפקדים שבתחום מסגרת הפקד יש מכנה משותף (בדרך כלל הם יהיו מערך של פקדים).
	CommandButton	זהו לחצן פקודה המבצע פעולה או פקודה כאשר המשתמש לוחץ עליו. כמתכנתים, אנו קובעים מהי הפעולה שתתבצע עקב הלחיצה.

CheckBox	פקד זה מציג אפשרויות של כן/לא (Yes/No), או אמת/שקר (True/False). כאשר קיימות בפני המשתמש כמה אפשרויות מסוג זה, הוא יכול לבחור ביותר מאפשרות אחת, או לחילופין באף לא אחת מהן.
OptionButton	לחצן אפשרויות זה מופיע בדרך כלל בקבוצה של מספר פקדים מסוגו (לרוב הם מהווים מערך פקדים), המאפשרים למשתמש לבחור באחד מהם. המשתמש יכול לבחור רק באחד מהפפקדים בקבוצה, ואין לו אפשרות לא לבחור כלל. מקובל לאגד קבוצת פקדים אלה בפקד Frame בתוספת כותרת מתאימה, המהווה חלק מה- Frame.
ComboBox	זהו פקד המשלב בתוכו שני פקדים, הן מבחינה ויזואלית והן מהבחינה הפונקציונלית. הפקדים המשולבים בו הם : TextBox ו-ListBox (ראה את הפקד הבא). מצד אחד יש למשתמש יכולת לשנות את הכתוב בתיבת הטקסט (בסגנון אחר של הפקד ניתן למנוע זאת מהמשתמש), ומצד שני הוא יכול לבחור פריט מהרשימה הנגללת.
ListBox	פקד זה מציג רשימת פריטים, שהמשתמש יכול לבחור באחד מהם.
HScrollBar **VScrollBar**	אלה הם סרגלי גלילה, אופקי ואנכי. סרגלי גלילה מוכרים לנו מסביבת Windows ותפקידם לאפשר טווח צפייה גדול יותר. בהמשך נלמד כי השימוש בפקדים אלה מגוון מאוד.
Timer	פקד זה מבצע פעולה מוגדרת בפרקי זמן קצובים. הפעולה ופרקי הזמן נקבעים על ידי המתכנת.
DriveListBox	מציג למשתמש את הכוננים המוגדרים אצלו במחשב ומאפשר לו לבחור מתוכם.
DirListBox	מציג למשתמש את רשימת התיקיות הקיימות במחשב ומאפשר לו לבחור תיקיה או נתיב.
FileListBox	מציג למשתמש את רשימת קבצים ומאפשר לו לבחור מתוכה קובץ רצוי.
Shape	פקד זה מאפשר להוסיף לטופס צורות גיאומטריות כמו ריבוע, מלבן, אליפסה ועיגול.

	Line	פקד זה מאפשר לצייר קו בטופס.
	Image	הפקד מציג תמונות מסוג bitmaps, icons, windows metafiles, gif ועוד. פקד זה שונה במקצת מהפקד PictureBox. על שוני זה נעמוד בהמשך. בנוסף, פקד זה יכול להתנהג כלחצן ולהגיב ללחיצה עליו.
	Data	פקד זה מאפשר להתקשר למסד נתונים (Database) קיים, ולאחזר ממנו נתונים אל הטופס.
	OLE	בעזרת פקד זה ניתן לשבץ אובייקטים מיישומי Windows שונים אל יישומי ויז'ואל בייסיק.

אל הפקדים הסטנדרטיים האלה ניתן להוסיף פקדים נוספים רבים. שני פקדים שימושיים חדשים לגירסה 6 הם **MonthView** ו-**DTPicker** שהם פקדי תאריכים. הפקדים דומים לפקדים אחרים, אלא ש-DTPicker הינו פקד בסגנון תיבה משולבת (ComboBox).

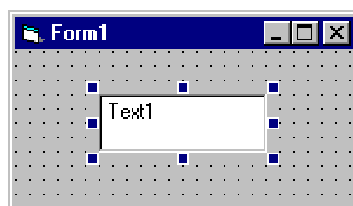
שימוש בפקדים

בסעיף זה נלמד כיצד מנצלים את הפקדים בפיתוח יישום ויז'ואל בייסיק, וכיצד יש להציב אותם על פני הטופס. בנוסף, נסביר גם את המאפיינים המייחדים כל פקד.

כל פקד, ללא תלות בסוגו, חייב לעבור את שני השלבים הבאים כדי שיופיעו בטופס:

1. לחיצה על הפקד ובחירתו מתוך ארגז הכלים (ToolBox).
2. תיחומו בשטח כלשהו על פני הטופס.

אופן התיחום נעשה באמצעות גרירת העכבר על פני הטופס. תנועת סמן העכבר מסמנת מלבן (שייעלם אחר כך) המגדיר את אזור הפקד בטופס. כאשר משחררים את העכבר בסיום הגרירה נקבע תחום הפקד בטופס. בשלב זה מופיעה סביב הפקד מסגרת שבה מסומנות 8 נקודות, המאפשרות להגדיל ולהקטין את הפקד (ראה תרשים 3.1).



תרשים 3.1

הנקודות שעל המסגרת קרויות גם **נקודות אחיזה** (Handles). העברת הפקד על פני הטופס ממקום למקום נעשית על ידי גרירתו באמצעות העכבר. לחיצה על מקום כלשהו בטופס, מחוץ לשטח הפקד, תבטל את ה"בחירה" בו ותעלים את המסגרת המקיפה אותו ואת נקודות האחיזה. לחיצה נוספת על הפקד, תחזיר את סימון הנקודות ותציין שהפקד "נבחר". ברגע שהפקד "נבחר", כל פעולה שנעשה תתייחס אליו, כפי שפעולות מתייחסות לטקסט שנבחר. אם לדוגמה, נלחץ על מקש Delete הפקד יימחק. בחירת מספר פקדים יחד תיעשה באחד משתי דרכים אלו:

1. בחירת הפקד הראשון, לחיצה ממושכת על מקש Shift או Ctrl, ובחירת פקדים נוספים.
2. גרירת סמן העכבר על פני הטופס (בדומה לסימון שטח עבור הפקד) ויצירת מלבן מקווקו תוחם. הדבר גורם לבחירת כל הפקדים (או אפילו קטע כלשהו מהם) שנמצאים באזור שסומן. במקרה זה, כל פעולה שתתבצע תחול על קבוצת הפקדים שנבחרה. לדוגמה, לחיצה על מקש Delete במקרה זה תמחק את כל הקבוצה.

עתה ניגש ללמוד כיצד פועלים עם הפקדים השונים.

א. PictureBox – תיבת תמונה

פקד זה מציג תמונה בגודלה האמיתי. אם שטח הפקד גדול מהגודל המקורי של התמונה נוכל לראות את התמונה בשלמותה. אם לתמונה יש מסגרת, נראה מרווח בין התמונה למסגרת, אלא אם נציב את הערך True למאפיין AutoSize. אולם, אם התמונה גדולה משטח הפקד, התמונה תחתך. יוצאי דופן במקרה זה הן תמונות מסוג Windows metafiles אשר מתאימות עצמן אוטומטית לגודל הפקד.

המאפיין **Picture** של הפקד מכיל את שם קובץ התמונה ואת הנתיב שלו.

ויז'ואל בייסיק שומרת את קובץ התמונה. כאשר מכינים ערכת התקנה ומפעילים את היישום במחשב אחר, עדיין נראה בפקד את התמונה ששיבצנו, למרות שבמחשב החדש אין קובץ תמונה דומה. ויז'ואל בייסיק שומרת את התמונה ו"לוקחת" אותה עימה אל קובץ ההרצה.

את קובץ התמונה אנו קובעים במאפיין Picture דרך חלון המאפיינים (Properties Window). ניתן לבצע זאת גם באופן דינמי במהלך התוכנית, אולם הטעינה תיעשה בעזרת קוד ולא דרך חלון המאפיינים. שורת קוד כזו תיראה כך:

```
PictureBox.Picture = LoadPicture("C:\Windowd\Bird.bmp")
```

הפונקציה **LoadPicture** מקבלת כפרמטר מחרוזת, המציינת את שם קובץ התמונה ונתיב שיוצג ב-PictureBox. ניתן לשלוח לפונקציה **משתנה מחרוזת** (String), המכיל את שם הקובץ ואת נתיבו. הפונקציה פונה לקובץ המתאים, טוענת אותו מהזיכרון ומציגה אותו בפקד.

שימוש נוסף ב-PictureBox הוא ביכולת הפקד הזה להכיל בתוכו פקדים אחרים. יכולת זו ניתן לנצל בבניית **שורת מצב** (Status Bar) שנוכל להציבה בתחתית הטופס, אם נציב בתוך הפקד PictureBox מספר פקדים מסוג Label. במצב שבו פקד Label מוכל בתוך פקד PictureBox, המאפיינים Top ו-Left של Label יתייחסו מעתה לפקד PictureBox, ולא לטופס.

כדי לשמור באופן קבוע על מיקומו של פקד PictureBox בתחתית הטופס, נקבע את ערך המאפיין Align שלו ל- Align Bottom - 2. זוהי אחת האפשרויות או הערכים של המאפיין Align (במקרה זה הערך השני).

ב. Label – תווית

פקד זה מציג טקסט על פני הטופס למטרת הודעה, הנחיה וכו'. המשתמש אינו יכול לשנות את הטקסט הכתוב בו. המאפיין העיקרי ב- Label הוא **Caption** (כותרת), או הטקסט אשר יוצג בו. השימוש במאפיין זה נעשה כך :

```
Label.Caption = "Enter your Name:"
```

אם הטקסט קבוע ב-Label במהלך ביצוע התוכנית, נקבע את ערכו במאפיין Caption שבחלון המאפיינים (Properties Window) ולא בשורת קוד. הסיבה היא שכך יותר נוח לעשות, ולא שאסור או אי-אפשר לעשות זאת.

שני מאפיינים נוספים, חשובים, מאפשרים שימוש בפקד בזמן ריצה. נניח שאנו רוצים להציג בעזרת Label את ציון המבחן לצד שם הסטודנט. בכל פעם שישתנה הטקסט ישתנה במקביל גודלו של הפקד מכיון שהשמות אינם שווים באורכם. תופעה זו מעוררת את השאלה, מהו השטח שנקצה לפקד? אם נקצה שטח קטן, שם סטודנט ארוך לא ייכנס בו ולחילופין, הקצאת שטח גדול תגרום לבזבוז מקום.

כאן נכנס לתמונה המאפיין **AutoSize**. מאפיין זה מאפשר לפקד לגדול או לקטון באופן אוטומטי על פי גודל הטקסט הכתוב בו. כדי שפעולה זו תתבצע ערך המאפיין צריך להיות True.

מאפיין דומה, אם כי מעט שונה, הוא **WordWrap**. קביעת ערך True במאפיין זה מאפשרת למשפט שבתוך הפקד לגלוש לשורה הבאה כאשר אורך הפקד אינו מספיק להכיל את המשפט בשלמותו (ראה תרשים 3.2). כדי שמאפיין WordWrap יאפשר גלישת שורות, חובה לקבוע את ערך המאפיין AutoSize ל-True.



תרשים 3.2

ג. TextBox – תיבת טקסט

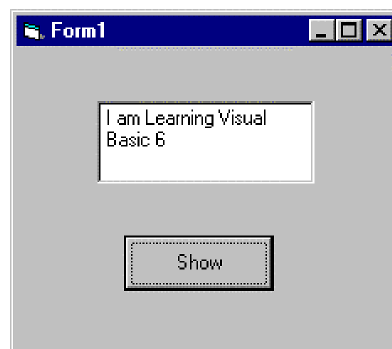
בפקד TextBox ניתן להציג טקסט, ובניגוד לקודמו - המשתמש יכול לערוך את הטקסט הכתוב בו ואף למחוק אותו. לפעמים נמצא את תיבת הטקסט ריקה ועל המשתמש למלא אותה בהתאם, כמו לדוגמה, כאשר אנו רוצים שהמשתמש ימלא את פרטיו כמו שם, כתובת וכד׳. המאפיין הנפוץ של פקד זה הוא Text. השימוש בו נעשה באופן הבא:

```
TextBox.Text = "Dany"
```

מאפיין חשוב נוסף הוא **MultiLine**. פקד TextBox רגיל מסוגל לקלוט אך ורק שורה אחת, ואף היא מוגבלת לאורך הפקד. מה יקרה אם נרצה להכניס כמה שורות? במקרה זה נשנה בחלון המאפיינים את ערך המאפיין MultiLine ל-True. במילים אחרות, נרשה ל-TextBox לקבל שורות. אך מה יקרה אם נרצה שורות רבות מעבר לקיבולת שטח הפקד? במקרה זה נפנה למאפיין **ScrollBars** ונבחר בערך המתאים.

באפשרותנו לבחור בסרגל גלילה אופקי, בסרגל אנכי, בשניהם יחד, או כלל לא. בכל מקרה עלינו לקבוע את הערך הדרוש. כאשר נבחר בסרגל גלילה, אנו יכולים לגלול את הטקסט שבתיבה אם הוא גולש מעבר ליכולת ההצגה של תיבת הטקסט TextBox. שימו לב! אין אלה פקדים של סרגלי גלילה, מדובר במאפיין של הפקד TextBox, אשר למעשה ממלא תפקיד דומה לפקד סרגל הגלילה.

נבצע אתנחתא קצרה לצורך תרגיל פשוט. מטרת התרגיל, ליצור טופס עם שני פקדים **TextBox** ו-**CommandButton**. בלחיצה על פקד הלחצן נקבל בתיבת הטקסט שתי שורות כמודגם בתרשים 3.3.



תרשים 3.3

לצורך ביצוע הפעולה נציב את הפקדים המתאימים על הטופס ונקבע להם את המאפיינים כמודגם בטבלה 3.1.

טבלה 3.1

ערך	מאפיין	פקד
txtVB (Empty) True	Name Text MultiLine	TextBox
cmdShow	Name	CommandButton

כעת נעבור למבט קוד של הטופס. בתרגיל שלנו, האירוע הוא Click של הפקד cmdShow. כדי לעבור באופן ישיר לשיגרה צריך להיות במבט עיצוב וללחוץ לחיצה כפולה על הפקד cmdShow המופיע בטופס. בחלון הקוד יופיעו באופן אוטומטי שורת ההצהרה ושורת הסיום של השיגרה. כל שנותר הוא לקבוע את ערך המאפיין Text של הפקד txtVB. בסיום, שורות הקוד ייראו כך:

```
Private Sub cmdShow_Click()
      txtVB.Text = "I am learning Visual Basic 6 "
End Sub
```

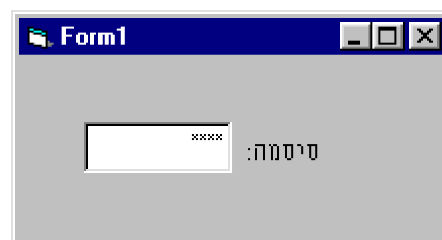
להרצת התוכנית הקש **F5**. המשפט "I am learning Visual Basic 6" יופיע בשתי שורות, בהנחה שאורך שטח הפקד אינו מכיל את כל המשפט בשורה אחת. להכנסת משפטים רבים ניעזר במאפיין ScroolBars.

הבה ונסקור שני מאפיינים נוספים:

PasswordChar. מוכרות לנו התוכניות בהן אין אנו רשאים להשתמש אלא-אם-כן אנו מקישים סיסמה. גם אם נרצה ל"הציץ" לבעל הסיסמה בזמן שהוא מקיש אותה, נכונה לנו הפתעה, כי במקום אותיות הסיסמה מופיעות כוכביות (****). בדיוק כך פועל המאפיין PasswordChar כאשר נקבע את ערכו ל- * (כוכבית). בכל פעם שנכתוב לתיבת הטקסט, לא תופענה האותיות שנקיש כי אם כוכביות.

MaxLength. מאפיין זה מגביל את מספר התווים שאנו יכולים להכניס לתיבת הטקסט. ערך המאפיין מציין את מספר התווים שמותר לכתוב. אם נחזור לתרגיל הקודם ונקבע למאפיין MaxLength את הערך 4, נראה בתוך תיבת הטקסט "I am" בלבד, כי רווח נחשב לתו.

ניתן לשלב את שני המאפיינים יחד. במקרה זה, קביעת ערך 4 למאפיין MaxLength והשמת * (כוכבית) במאפיין PasswordChar יבנו לנו תיבת סיסמה מושלמת, כדוגמת התרשים הבא:



תרשים 3.4

לסיום הדיון בפקד **TextBox** הנה טיפ קטן. כאשר המשתמש כותב לתוך תיבת טקסט ריקה, הכל טוב ויפה. אך מה קורה כשהוא חוזר בו ורוצה לכתוב משפט או מילה אחרת? אין זה נוח למחוק תו אחר תו. מי שמכיר את סביבת העבודה בחלונות יודע, כי במצב כזה התיבה נבחרת ("נצבעת") באופן אוטומטי ובהקשת תו חדש כל הכתוב קודם נמחק. כל שנותר הוא לכתוב את המילה החדשה בלי לעמול במחיקת קודמתה. בוויז'ואל בייסיק הדבר נעשה בשתי שורות קוד בלבד.

נחשוב יחד, מהו האירוע המתאים לכתיבת שורות הקוד. המשתמש מצפה, כי בכל פעם שהוא יתמקד בתיבת הטקסט, המילים הכתובות בה יסומנו. במילים אחרות, בכל פעם שתיבת הטקסט מקבלת "מיקוד" (focus) מהמשתמש – תתבצע פעולת הצביעה. מעיון בתיבת השגרות בחלון הקוד נמצא כי האירוע העונה לדרישה זו הוא **GotFocus**. שלב ראשון, אם כן, יהיה פנייה למבט קוד של הטופס, בחירה בפקד TextBox ובשיגרה GotFocus. מה שנקבל יהיה:

```
Private Sub Text1_GotFocus()
End Sub
```

לפקד TextBox יש שני מאפיינים שאינם מופיעים בחלון המאפיינים, אלא אך ורק בשורות הקוד. נזכור בהזדמנות זו שיש מאפיינים שניתן לקבוע את ערכם אך ורק בשורת קוד (Run Time). נכיר שניים ממאפיינים אלה:

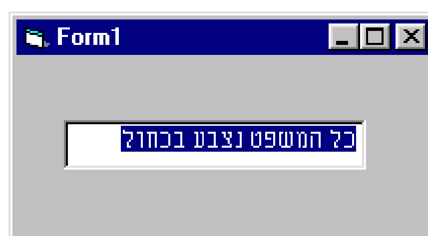
SelStart. מאפיין זו מורה לסמן, המהבהב בתוך תיבת הטקסט, באיזה מקום לעמוד. באופן טבעי הסמן עומד בסוף המילה. בעזרת מאפיין זה נוכל לשנות את מיקומו בתיבה. אם נרצה לשנות את מיקום הסמן לראש המילה או המשפט, נקבע את ערכו ל-0 (אפס). שורת קוד תיראה כך:

```
Text1.SelStart = 0
```

SelLength. מאפיין זו מסמן את מספר התווים בתיבת הטקסט כפי שהוגדר לו, החל מהמקום בו עומד הסמן והלאה. אם נרצה לצבוע 4 תווים ממקום הסמן והלאה, נכתוב:

```
Text1.SelLength = 4
```

שילוב של השניים, מיקום הסמן בראש התיבה (בעזרת SelStart) וסימון מספר התווים הרשומים בתיבה (בעזרת SelLength), יניב את התוצאה שרצינו – צביעת המשפט, כדי לחסוך מהמשתמש את מחיקתו. לדוגמה:



תרשים 3.5

עדיין לא סיימנו. אנו יודעים למקם את הסמן בראש המשפט. אנו גם יודעים לצבוע ממקום הסמן והלאה את מספר התווים שנרצה לצבוע, אך מהו אותו מספר? כיצד נדע מהו אורך המילה שהמשתמש יקליד? כיצד נדע לקבוע כמה תווים לצבוע?

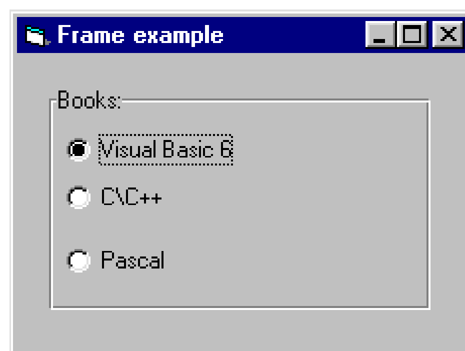
לצורך כך ניעזר בפונקציה Len(). הפונקציה Len() מקבלת כפרמטר מחרוזת מסוימת ומחזירה את מספר התווים שנמצאים בה. המאפיין המייצג את המחרוזת הכתובה בתיבת הטקסט הוא Text. קריאה לפונקציה Len() עם המאפיין Text כפרמטר תפתור לנו את הבעיה – מציאת אורך המחרוזת, או במילים אחרות: את מספר התווים הכתובים בפועל בתיבת הטקסט.

מכל האמור עד כאן למדנו שכדי לסמן מחרוזת (במידה והיא קיימת) בתיבת טקסט, עלינו לרשום את שורות הקוד הבאות:

```
Private Sub Text1_GotFocus()
      Text1.SelStart = 0
      Text1.SelLength = Len(Text1.Text)
End Sub
```

ד. Frame – מסגרת

השימוש העיקרי של הפקד הוא, כפי שכבר למדנו, ליצור מסגרת סביב פקדים בעלי מכנה משותף, ולהציב להם כותרת בעלת משמעות. מבחינה פונקציונלית (שהיא הבחינה החשובה יותר), ניתן לאגד בתוכו קבוצת פקדים כמערך, כפי שנלמד בסעיף הבא. השימוש הנפוץ במצב זה הוא, הצבת קבוצת פקדים מסוג OptionButton בתוך מסגרת המאפשרת בחירת אפשרות אחת מתוך מספר אפשרויות (ראה תרשים 3.6).



תרשים 3.6

אנו קובעים באמצעות המאפיין Caption את אשר ייכתב בכותרת.

ה. CommandButton – לחצן פקודה

לחיצה על הפקד מאפשרת לבצע פעולה מוגדרת, כאשר לוחצים עליו. הכיתוב שעל הלחצן מציין זאת, לנוחות המשתמש. הכיתוב נקבע באמצעות המאפיין Caption. אם לדוגמה, נקבע כי

```
CommandButton.Caption = "Exit"
```

נצפה, כי בלחיצה על הלחצן נצא מהמסך הפעיל, או נצא מהתוכנית.

ישנה אפשרות לשייך "מקש חם" ללחצן. אם נצמיד את הסימן "&" לאחת מאותיות הכיתוב (Caption) של הלחצן, נגרום להצגת קו תחתון תחת אותה אות. במקרה הבא:

```
CommandButton.Caption = "&Exit"
```

יופיע קו תחתון תחת האות E, כמודגם בתרשים 3.7:



תרשים 3.7

במקרה זה, לחיצה על מקש Alt עם האות המסומנת בקו תחתון, כמוה כלחיצה בעכבר על הלחצן, או הקשת Enter כאשר המיקוד הוא בלחצן. בדוגמה שלנו, הקשת Alt+E תפעיל את האירוע ()CommandButton_Click.

שימוש נוסף בפקד הוא, הפיכתו ללחצן גרפי, אשר מוצגת בו תמונה במקום כיתוב. הקבצת מספר לחצנים גרפיים בראש הטופס יכולה להוות סרגל כלים מאולתר (על בניית סרגל מקצועי נלמד בפרק 17).

כיצד הופכים לחצן פשוט ללחצן גרפי עם תמונה?

לצורך כך נבצע את שני השלבים הבאים:

1. נמחק את תוכן המאפיין **Caption** של הלחצן. אם נרצה שבתחתית התמונה יוצג כיתוב מתאים, עלינו להציב טקסט במאפיין זה. למאפיין Style יש שני ערכים: **Standard** מורה על לחצן רגיל עם כיתוב על גביו; הערך **Graphical**, שמעניין אותנו. בערך זה נבחר כדי להפוך את הלחצן ללחצן גרפי המציג תמונה.
2. במאפיין **Picture** נציין את שם קובץ התמונה ונתיבו, אשר יוצג בתמונת הלחצן.

שאר הפעולות הנעשות עם הלחצן, כבחירת האירוע וכתיבת הקוד בהתאם, זהות בשני סוגי הלחצנים.

שני מאפיינים חשובים נוספים של פקד זה הם: **Cancel** ו-**Default**. למאפיינים אלה נציב את הערכים True או False. True מחיל את הפעולה שהמאפיין אחראי לה; False מבטל פעולה זו. כאשר המשתמש מקיש על מקש Esc, הוא כאילו לחץ על הלחצן (או הפקד CommandButton) בו ערך המאפיין Cancel הוא True, למרות שהפוקוס הוא לא על הלחצן אלא על פקד אחר. באותו אופן פועל המאפיין Default, אך המקש במקרה זה הוא Enter. חשוב לציין, שכאשר יש מספר לחצנים על הטופס,

רק אחד מהם יכול לקבל ערך True באחד מן המאפיינים או בשניהם. לא בהכרח שלחצן אחד יפעיל את שתי הפעולות גם יחד. לחצן אחד יכול להשתמש במאפיין Cancel ואחר – ב-Default.

ו. CheckBox – תיבת סימון

פקד זה מאפשר למשתמש לבחור באפשרות מסוג כן/לא (Yes/No) או אמת/שקר (True/False). במילים אחרות, סימון ☑ בתיבת הסימון כמוה כבחירת הפעולה שהיא מייצגת. ביטול הסימון והשארת התיבה ריקה, מבטלת את הבחירה. כיצד נוכל, אנו המתכנתים, לדעת אם המשתמש לחץ על התיבה וביקש לבחור בפעולה שהיא מייצגת, או שמטרת לחיצתו היתה ביטול הבחירה? לצורך זה עומד לרשותנו המאפיין Value, שערכו מורה לנו על בחירת המשתמש. כאשר ערך המאפיין הוא 0 (אפס, Unchecked), הדבר מצביע על כך שהמשתמש ביטל את בחירתו והתיבה נקייה מסימון; כאשר ערך המאפיין הוא 1 (Checked) הדבר מורה שהתיבה מסומנת.

בשורת קוד, ניתן להשתמש לחילופין בקבוע vbChecked המייצג את הערך 1. שתי שורות הקוד הבאות זהות לחלוטין:

```
Check1.Value = 1
Check1.Value = vbChecked
```

כאשר מציבים על פני הטופס מספר פקדים מסוג זה, ניטיב לעשות אם נבנה אותם כמערך.

כמקובל בסביבת עבודה חלונאית, כאשר המשתמש רואה לפניו מספר אפשרויות מסוג CheckBox, הוא מבין שבאפשרותו לסמן (או לבחור) יותר מתיבה אחת, או לחילופין אף לא אחת מהן.

ז. OptionButton – לחצן אפשרויות

הפקד OptionButton דומה מאוד באופן פעולתו לפקד CheckBox. גם הוא נמצא בדרך כלל בקבוצת אפשרויות, רצוי מערך. בדרך כלל נקבץ את הפקדים בתוך מסגרת (הפקד Frame) וניתן לה כותרת מתאימה (ראה שוב תרשים 3.6). המאפיין Value ממלא פונקציה דומה לזו של תיבת סימון (CheckBox), אך הפעם הערכים הם True או False: True מורה על בחירה ו-False על ביטול בחירה. יש הבדל בין הפקדים אשר נובע מיכולת המשתמש לבחור. במילים פשוטות, כאשר מונחות לפני המשתמש מספר אפשרויות מסוג OptionButton, חובה עליו לבחור באפשרות אחת, ורק אחת, בלבד.

ח. ComboBox – תיבה משולבת

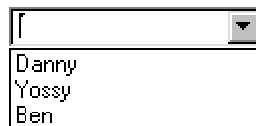
יש דמיון רב בין פקד זה והפקד ListBox שיוסבר בהמשך.

התיבה המשולבת ComboBox היא שילוב פונקציונלי של הפקדים TextBox ו-ListBox. כתיבת טקסט, TextBox, ניתן להציג בה טקסט או לערוך בה טקסט

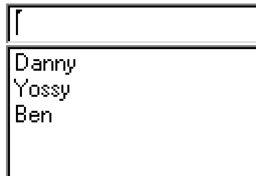
קיים; וכתיבת רשימה, ListBox, היא מאפשרת לבחור פריט מתוך רשימת אפשרויות קיימת.

לתיבה משולבת יש מאפיין בשם Style המקבל שלושה ערכים (0-2). כל ערך מייצג סגנון שונה של התיבה, מבחינה ויזואלית ומבחינה פונקציונלית. הבה נכיר אותם:

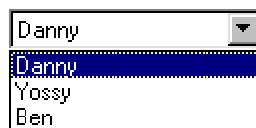
Style = 0 (ברירת המחדל) מייצג תיבת טקסט קופצת (pop-up). כמו שתואר קודם, ניתן לכתוב בה, או לבחור מתוך הרשימה הנפתחת.



Style = 1 הוא סגנון פשוט, המציג תיבת טקסט על מאפייניה. מתחתיה מוצגת רשימה גלויה וקבועה (בפורמט ListBox). לדוגמה,



Style = 2 דומה במבנהו לסגנון הראשון. ההבדל החשוב הוא, שבסגנון זה אין אפשרות לכתוב לתוך התיבה. המשתמש יכול לבחור אחת מהאפשרויות בלבד, ללא יכולת שינוי ועריכה. לדוגמה,



פקד חדש בוויז'ואל בייסיק 6 הוא פקד ComboBox עם תמונות. פקד זה נקרא **ImageCombo** וצורת הסמל שלו היא . מלבד רשימת האפשרויות שהוא מכיל, ניתן להצמיד תמונה לכל אפשרות.

ט. ListBox – תיבת רשימה

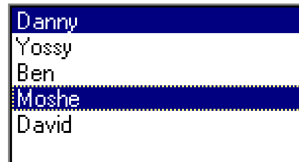
ListBox היא תיבה פתוחה המכילה אפשרויות שונות לבחירה (אם גודל הפקד לא מאפשר לראות את כל הפריטים ברשימה, באופן אוטומטי מופיע סרגל גלילה). עדיף להקצות מלכתחילה שטח מספיק גדול לפקד, היות ולתיבה זו (בניגוד ל- ComboBox) אין יכולת להיפתח ולהיסגר, גודלה קבוע!

המשתמש יכול לבחור באחד מהפריטים המוצגים ברשימה ללא יכולת עריכה. במידה ונרצה לאפשר למשתמש יכולת בחירה מרובה, ניעזר במאפיין **MultiSelect**. למאפיין זה שלושה ערכים:

None - 0 (ברירת המחדל), מאפשר בחירת פריט אחד בלבד.

Simple - 1, מאפשר בחירה מרובה. אופן הבחירה נעשה בלחיצת עכבר על כל אחד מהפריטים הרצויים.

לדוגמה:

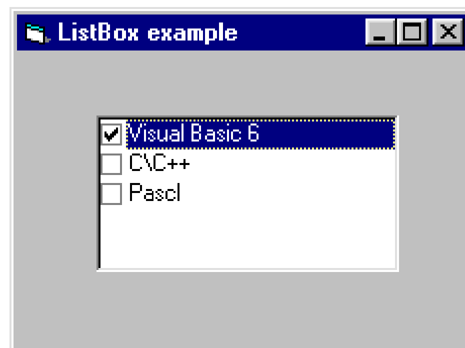


Extended - 2, מאפשר בחירה מרובה. אופן הבחירה דומה במקרה זה לאופן בחירת קבצים בסייר Windows. הבחירה נעשית בעזרת Shift ו-Ctrl לבחירה רציפה ושאינה רציפה בהתאמה.

מאפיין מעניין נוסף של פקד זה הוא **Columns**, אשר מקבל ערך מספרי המייצג את מספר העמודות שיופיעו בתיבת הרשימה. לדוגמה, אם נקבע את הערך 4 יופיעו הפריטים בארבעה טורים. כאשר שטח הפקד קטן יחסית ומספר הפריטים רב, יופיע באופן אוטומטי סרגל גלילה אופקי. כל תזוזה של הסרגל תציג פריטים חדשים במסגרת 4 טורים. לדוגמה,

Danny	Moshe	Gad
Yossy	David	Shlomo
Ben	Avi	

ניתן להציג את תיבת הרשימה בסגנון נוסף, שאפשר לקבוע אותו בעזרת המאפיין **Style** של הפקד. בחירה בערך **Checkbox** של המאפיין **Style** תגרום להצגת תיבת רשימה בסגנון הבא:



3.8 תרשים

המשותף לשני הפקדים: בשניהם אנו מוסיפים ומורידים פריטים מהרשימה באופן דומה. המאפיינים הבאים מתייחסים לזמן ריצה - פעולת התוכנית - ועל כן, נמצא אותם אך ורק בשורות הקוד, ולא בחלון המאפיינים.

- **AddItem** - מוסיף פריטים לרשימה. בשורת קוד הדבר ייראה כך:

```
ComboBox / ListBox.AddItem Value
```

לדוגמה:

```
List1.AddItem "Israel"
List1.AddItem "USA"
List1.AddItem "France"
```

- **Clear** - מנקה את תוכן הרשימה. בשורת קוד הדבר ייראה כך:

```
ComboBox / List1.Clear
```

- הוספת פריט במקום ספציפי ברשימה:

```
ComboBox / List1.AddItem ″Japan″ , 0
```

במקרה זה הפריט ״Japan״ הוצב באינדקס 0, שהוא המקום הראשון ברשימה.

- הסרת פריט מסוים מהרשימה תיעשה באופן הבא:

```
ComboBox / List1.RemoveItem Index
```

ולדוגמה,

```
List1.RemoveItem 2
```

כאן יוסר הפריט השלישי ברשימה; מכיון שמספרי האינדקס מתחילים ב-0, הפריט השלישי הוא בעל אינדקס 2.

- הגישה לפריט מסוים ברשימה תיעשה באופן הבא:

```
ComboBox / List1.List(Index)
```

לדוגמה, גישה לפריט השלישי ברשימה:

```
Text1.Text = List1.List(2)
```

- **ListCount** מציין את מספר הפריטים הנמצאים ברשימה.

```
ComboBox / List1.ListCount
```

בדוגמה זו, תיבת הטקסט מקבלת כערך את מספר הפריטים ברשימה.

```
Text1.Text = List1.ListCount
```

- המספר הסידורי של האיבר הנוכחי (הנבחר) מתקבל באמצעות **ListIndex**.

```
ComboBox / List1.ListIndex
```

מכיון שמספרי האינדקס מתחילים באפס, טווח האינדקסים של כל הרשימה נע מאפס ועד ListCount-1 (כשיש 4 פריטים, האינדקס של הפריט האחרון הוא 3).

י. ScrollBar – פס גלילה

יש שני פקדים מסוג זה, פס גלילה אנכי ופס גלילה אופקי. מכיון ששניהם מתפקדים באופן דומה, נתייחס אל פס הגלילה באופן כללי. פס הגלילה - כפקד - אינו תלוי בפס הגלילה שעל הטופס, או באלה שאנו מכירים ב-ComboBox, ListBox ועוד. לפקד פס הגלילה תפקיד נוסף. מקובל להשתמש בו ככלי למדידה והערכה, כדוגמת סקלת Sound Volume לשינוי עוצמות הקול. גלילה כלפי מעלה תגביר את עוצמת הקול, וכן להיפך. ניתן גם להשתמש בפקד זה כציון לזמן שנדרש לתהליך מסוים להתרחש. ככל שפס הגלילה מתקרב אל סופו (אל קצה הפס), המשתמש יודע כי התהליך עומד להסתיים, וכשפס הגלילה באמצע הסרגל, מעיד הדבר כי הוא נמצא באמצע התהליך. באופן דומה נוכל לקבל אינדיקציה בכל זמן ממשך התהליך.

המאפיינים הנפוצים של סרגל הגלילה הם : Min, Max ו-Value.

המאפיין Min קובע את הערך המינימלי של הסרגל. אם נחזור לדוגמת ה- Sound Volume, מאפיין זו יורה על גובה צליל הנמוך ביותר. באופן דומה, Max יקבל את הערך הגבוה ביותר. המאפיין Value מאפשר לקרוא בכל רגע נתון באיזו נקודה בפס נמצאת הגררה שעליו; במילים אחרות, מהו הערך הנוכחי בפס הגלילה. הפנייה למאפיין תיעשה באופן המוכר לנו :

```
Label.Caption = Vscroll.Value
```

בדוגמה זו, הפקד Label מציג בטופס את הערך הנוכחי של הפס.

שני אירועים מאפשרים לקבל משוב מפס הגלילה על השינוי שחל בו, ולהגיב אם דרוש. את האירועים, כבר למדנו, אנו מוצאים במבט קוד של הטופס, בתיבת השגרות (התיבה הימנית).

האירוע הראשון הוא **Change**. אירוע זה מתרחש לאחר שחל שינוי בפס הגלילה. תפקידנו כמתכנתים, לקבוע מה לעשות במקרה זה, או מהן שורות הקוד המתאימות אשר יבוצעו בכל שינוי שיחול בפס.

האירוע השני הוא **Scroll**. אירוע זה מתרחש בזמן שהגררה נעה בתוך פס הגלילה. שימו לב, כי האירוע לא מתרחש בזמן שלוחצים על חיצי הגלילה, אלא בזמן תנועת פס הגלילה בלבד!

יא. Timer – מונה

באמצעות פקד זה ניתן לבצע פעולה מסוימת החוזרת על עצמה בכל פרק זמן רצוי. הפקד מונח על גבי הטופס בזמן עיצוב אך אינו נראה בזמן ריצה, רק הפונקציונליות שלו פועלת, ולכן אין חשיבות למיקומו בטופס, העיקר שיופיע שם. המאפיין העיקרי של הפקד הוא **Interval**, הקובע את מרווחי הזמן שבהם תופעל השיגרה שמאחורי הפקד. היחידות ב-Interval בנויות כך, שכל 1000 יחידות מציינות שנייה אחת. אם, לדוגמה, נרצה לקבוע את תדירות הופעת השיגרה ל-3 שניות, נקבע את ערך המאפיין Interval ל-3000. השיגרה העומדת מאחורי הפקד היא **Timer**, והיא גם היחידה שקיימת לפקד זה.

יב. DriveListBox – תיבת רשימת כוננים

פקד זה מאפשר להציג את כל הכוננים, לדפדף ביניהם ולבחור בכונן הרצוי. המאפיין **Drive**, שנמצא בשורות קוד בלבד, מקבל או מציג את ערך הבחירה.

יג. DirListBox – תיבת רשימת תיקיות

פקד זה ממלא פונקציה דומה לקודמו. השוני הוא בכך שפקד זה מציג את רשימת התיקיות. המאפיין **Path** מקבל או מציג את ערך הבחירה.

יד. FileListBox – תיבת רשימת קבצים

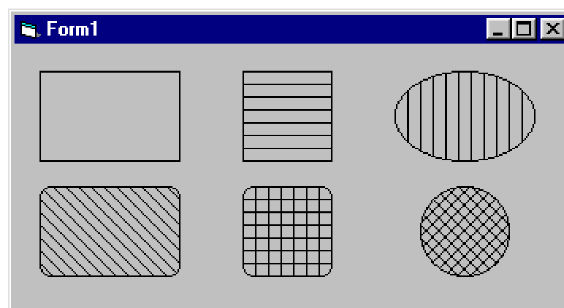
פקד זה מאפשר לצפות ברשימת הקבצים בתיקיה שנבחרה. המאפיין המאפשר לצפות בשם קובץ רצוי הוא **filename**. המאפיין **Path** מציג את הנתיב לקובץ, ללא שם הקובץ. בשילוב שני המאפיינים ובעזרת אופרטור השרשור &, נקבל את שם הקובץ ואת נתיבו. חשוב לא לשכוח לשרשר גם את התו "\" המפריד בין שם הקובץ לנתיבו, כדוגמת השורה הבאה:

```
File1.Path & "\" & File1.filename
```

טו. Shape – צורה

הפקד **Shape** מאפשר ליצור צורות גיאומטריות שונות על פני הטופס. שני המאפיינים העיקרים בפקד הם **Shape** - הקובע את הצורה הגיאומטרית, ו- **FillStyle** - הקובע את סוג המילוי של הצורה.

המאפיין Shape מקבל שישה ערכים המייצגים שש צורות שונות: מלבן, ריבוע, אליפסה, עיגול, מלבן בעל פינות מעוגלות וריבוע בעל פינות מעוגלות. המאפיין FillStyle מאפשר לבחור באחד משמונת סוגי מילוי: אחיד, קווים אלכסונים ועוד (ראה תרשים 3.9).



תרשים 3.9

טז. Line – קו

הפקד **Line** מאפשר לצייר קו על פני הטופס. ניתן לצייר קווים רבים, אולם כל קו הוא פקד בפני עצמו. ארבעת המאפיינים האלה - X1, Y1, X2, Y2 - מקבלים ערכים מספריים הקובעים את שתי הקואורדינטות של קצות הקו, והקו נמתח ביניהן. יש לשים לב, כי בניגוד למה שאנו נוהגים בשרטוט על נייר, כאן ציר Y עולה בערכו ככל שאנו **יורדים** בטופס. במילים אחרות, נקודת האפס של ציר Y מתחילה בראש הטופס. ציר X נשאר כמות שהוא בציר המספרים הרגיל, משמאל לימין. נקודת האפס 0,0 היא הנקודה השמאלית העליונה של הטופס, והנקודה הגבוהה ביותר היא הנקודה הימנית התחתונה.

יז. Image – דמות / תמונה

פקד **Image** דומה מאוד לפקד **Picture**, כי שניהם מציגים תמונות. ההבדל העיקרי הוא בכך שלפקד Image יש מאפיין **Stretch** המקבל ערכי True/False ומאפשר לשנות את גודל התמונה שנבחרה. אם הגודל המקורי של התמונה שנבחרה קטן או גדול מהשטח שהוקצה לפקד, נוכל לשנות את גודל התמונה ולהתאים אותה לגודל הרצוי לנו, בתנאי שנציין תחילה את הערך True במאפיין Stretch.

המאפיין **Picture** של פקד זה מקבלת כערך את שם קובץ התמונה ואת נתיבו.

היבט נוסף שכדאי לדעת הוא, שלפקד Image יש אירוע **Click**; כלומר, כאשר לוחצים על הפקד אפשר לגרום לביצוע האירוע Click (הדבר דומה ללחצן גרפי). אולם, בניגוד ללחצן (פקד CommandButton) הפקד Image לא ייראה "לחוץ".

יח. Data – נתונים

הפקד Data הוא בעל עוצמה רבה. בעזרתו ניתן להתקשר למסד נתונים קיים ולהציג את כל המידע שבו, בלא שימוש אפילו בשורת קוד אחת. החיסרון של הפקד בכך שהוא אינו נוח לעבודה מבחינת הנדסת אנוש. לכן, כשכותבים יישום המקושר למסד נתונים, מיישמים אותו בדרך כלל באמצעות קוד ללא שימוש בפקד Data. כך יש אמנם יותר שורות קוד, אך התוכנית, ובעיקר ממשק המשתמש ואפשרויות השימוש בו, טובים לעין ערוך. פרק 16 יוקדש לניהול מבנה נתונים באמצעות קוד.

יט. OLE – קישור והטבעת אובייקטים

באמצעות הפקד OLE ניתן לשבץ אובייקטים מיישומים אחרים לתוך יישומי ויז'ואל בייסיק.

מערכי פקדים

במהלך בניית ממשק המשתמש נוכל למצוא שאנו משתמשים מספר פעמים בסוג מסוים של פקד, ואפילו בטופס אחד. במצב זה נבנה בדרך כלל מערך פקדים, למרות שאין חובה ליצור מערך כזה. ניקח לדוגמה טופס, שבו אנו מציבים מספר פקדים מסוג TextBox לקליטת שם, כתובת, טלפון וכו׳, ולכל פקד ניתן את השם המיוחד לו על פי תפקידו. במקרה זה לא רצוי לתת לכל הפקדים שם אחד המייצג את המערך. במצב זה הבלבול יהיה גדול, כי כיצד נזכור שאינדקס 0 מייצג שם, 1 - כתובת, 2 - טלפון וכו׳. במקרה זה קל יותר לזכור ש-txtName, txtAddress ו- txtTelephone מתייחסים לשם, כתובת וטלפון בהתאמה.

מתי נשתמש במערך של פקדים?

נשתמש במערך של פקדים כאשר הפקד מבצע פונקציה דומה בכל המופעים שלו, ואין חשיבות לשם החוזר על עצמו. ניתן לומר, כי עדיף היה לתת לפקד את אותו שם כל הזמן, אך מכיון שפעולה זו לא תיתכן, אנו מגדירים מערך. כדוגמה ניתן לקחת את הפקד OptionButton. לפני המשתמש עומדת רשימת אפשרויות שממנה הוא אמור לבחור. נניח שרשימה זו מייצגת מספר מנות עיקריות לארוחת צהריים. במקרה זה, ניטיב לעשות, אם נקרא לפקד בשם optFood וכל פריט המייצג מנה עיקרית יהיה חלק ממערך. במקרה זה, הפתרון של הגדרת פקד נפרד לכל מנה (פקד בשר, פקד דג וכדי) יכול להיות מורכב ומבלבל יותר.

בניית המערך

בניית המערך הינה פעולה פשוטה ביותר. תחילה יוצרים את הפקד שרוצים להפוך למערך, וקובעים לו את המאפיינים הרצויים. לחיצה ימנית על הפקד תציג תפריט מקוצר, אשר אחת האפשרויות בו היא **Copy**. נבחר באפשרות זו כדי להעתיק (למעשה, לשכפל) את הפקד. לאחר העתקה זו (ללוח, ClipBoard) נבצע הדבקה אל הטופס. לחיצה ימנית נוספת תציג שוב את אותו תפריט, אך הפעם נבחר באפשרות **Paste**. בשלב זה תוצג שאלה, האם אנו מעוניינים שהפקד החדש שיווצר יהיה חלק ממערך, או לא. על שאלה זו נענה בחיוב, וכך ניצור את הפריט השני במערך (הפקד המקורי הוא הפריט הראשון במערך). מכאן והלאה, בכל פעם שנחזור על פעולת ההדבקה (Paste) יתוסף פריט נוסף למערך. אם תהיתם כי בכל פעם שאתם מבצעים את פעולת ההדבקה לא קורה שום דבר, אל דאגה. בכל פעם שמבצעים פעולת Paste, ויז׳ואל בייסיק מדביקה את הפקד החדש בפינה השמאלית-עליונה של הטופס (או של ה-Frame, אם בו יצרנו את המערך). בדרך זו, בכל הדבקה הפקדים מוצבים בפינה השמאלית העליונה, ועלינו לגרור אותם אל המקום הרצוי בטופס. חשוב לשים לב לכך שאנו גוררים אותם לפי סדרם הכרונולוגי במערך.

כיצד יודעים מהו הסדר?

אם נתבונן במאפיינים של כל פריטי המערך, נראה שהם זהים בכל המאפיינים מלבד שניים. הראשון הוא **Index**. המאפיין Index מקבל ערכים מספריים המייצגים את סדר הפקדים במערך. הפקד הראשון יקבל את הערך 0, השני - 1, השלישי - 2 וכו'. כך גם לגבי שם הפקד, המאפיין **Name**. בדוגמה שלנו השם המקורי של הפקד היה optFood. מרגע שנבנה המערך מצורף לשם הפקד גם מיקומו במערך, ובמילים אחרות - מספר האינדקס שלו. בדוגמה שלנו, optFood (0) - מתייחס לפקד הראשון, optFood (1) - לפקד השני, optFood (2) - לפקד השלישי וכן הלאה.

השימוש במערך בכתיבת קוד

נמשיך בדוגמה שהצגנו בסעיף הקודם ונניח שבנינו מערך פקדים מסוג OptionButton. נניח שבמערך יש שלושה פריטים המייצגים שלוש אפשרויות למנה עיקרית בארוחת צהריים במסעדה. פריט ראשון (Index=0) הוא דג, פריט שני (Index=1) עוף ופריט שלישי (Index=2) בשר. בנוסף קיים בטופס פקד lblPrice מסוג Label, המציג למשתמש את מחיר המנה העיקרית שבחר (ראה תרשים 3.10).



תרשים 3.10

המשתמש שלנו העדיף עוף לארוחת צהריים, והיה רוצה לדעת מה המחיר. הוא לחץ על האפשרות השנייה וחיכה לתשובה. עלינו כמתכנתים לדאוג להופעת התשובה הנכונה.

כיצד נעשה זאת?

בלחיצה על אחת האפשרויות, המשתמש הפעיל את האירוע **Click**. אם כן, עלינו לכתוב את שורות הקוד המורות לתוכנית כיצד להגיב בעת התרחשות אירוע זה. ניגש לחלון הקוד, נבחר בפקד optFood ובאירוע Click. כמו תמיד, השורה הראשונה

והשורה האחרונה של השיגרה נכתבות אוטומטית. הפעם יש שינוי קטן, השיגרה מקבלת פרמטר, כפי שנראה להלן:

```
Private Sub optFood_Click(Index As Integer)

End Sub
```

מהי סיבת השוני, ומהיכן יודעת השיגרה שעליה לקבל את Index כפרמטר?

התשובה פשוטה. זוכרים מה מיוחד בפקד optFood? הוא מערך! ומכיון שכך, כאשר נלחץ עליו ונפעיל את האירוע Click, לא די בכך שנכריז שהופעל אירוע לחיצה על הפקד optFood, אלא צריך גם לדעת על איזה פריט בתוך המערך המשתמש לחץ!

כזכור, יש לנו שלוש מנות עיקריות (או פריטים). כיצד נדע האם המשתמש בחר בדג, בעוף או בבשר? זהו למעשה תפקיד המשתנה Index בשיגרה. הוא מכיל את ערך האינדקס של האפשרות שנבחרה. אם Index=0 - פירוש הדבר שנבחרה האפשרות הראשונה, או במילים אחרות המשתמש בחר כמנה עיקרית בדג. אם Index=1 המשתמש בחר באפשרות השנייה המוצעת, וכך הלאה.

על פי קו מחשבה זה אפשר להרכיב את שורות הקוד כדי לבדוק מהי האפשרות שנבחרה על ידי המשתמש, ועל פיה נוכל להציג את הודעת התשלום. נניח שעבור מנת דג התשלום הוא 50 ש"ח, לעוף 30 ש"ח ולבשר 40 ש"ח. שורות הקוד שלנו, על פי זה, ייראו כך:

```
Private Sub optFood_Click(Index As Integer)
  Select Case Index
    Case 0
      lblPrice.Caption = 50
    Case 1
      lblPrice.Caption = 30
    Case 2
      lblPrice.Caption = 40
  End Select
End Sub
```

את מבנה ההחלטה **Select Case** נכיר בהמשך (פרק 8); עתה רק נאמר, כי הוא עוזר לנו בבירור ערך המשתנה Index. אם ערכו 0 (אפס), פירוש הדבר שנבחרה האפשרות הראשונה והודעת התשלום היא 50 ש"ח בהתאם. כך ננהג גם בשתי האפשרויות האחרות.

יש אפשרות להוסיף פריט למערך הפקדים בזמן ריצה. פעולה זו שימושית כאשר לא ידוע לנו מראש מספר הפריטים הכללי במערך. במקרה זה נוסיף (או נסיר), באופן דינמי, בזמן ריצה, פריט כלשהו. הוספת פריט מתבצעת באופן הבא:

```
Load ObjectName (Index)
```

בהמשך לדוגמת המסעדה, נניח שברצוננו להוסיף אפשרות למנה עיקרית. נעשה זאת כך:

```
Load optFood (3)
```

לחילופין, כדי להסיר פריט מהמערך בזמן ריצה, נשתמש בתחביר הבא:

```
Unload ObjectName (Index)
```

ולדוגמה:

```
Unload optFood (3)
```

חשוב לזכור, שהוספת פריט באופן דינמי מחייבת הצבה של לפחות פריט אחד מאותו פקד על פני הטופס, בזמן עיצוב. בדרך כלל פריט זה יקבל ערך Index=0, ובזמן ריצה נוסיף פריטים שונים עם אינדקסים בסדר עולה. אם אין אנו מעוניינים לראות את הפקד הראשון בזמן הריצה, נשתמש במאפיין **Visible** ונקבע את ערכו ל-False. עם זאת, פריט ראשוני זה במערך חייב להופיע בזמן העיצוב, כדי לאפשר הוספה דינמית בזמן ריצה של פריטים נוספים.

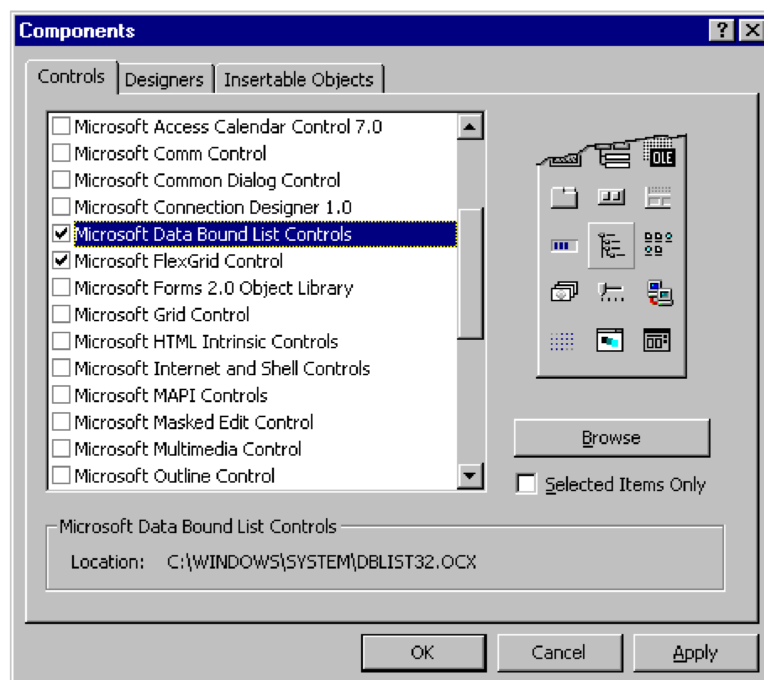
הוספת פקד לסרגל הכלים

ארגז הכלים מכיל ערכה סטנדרטית של הפקדים השימושיים ביותר, שבאמצעותם נעשית רוב עבודת הפיתוח של יישומים. עם זאת, ישנם פקדים שימושיים נוספים, שאינם מופיעים במתכונת הסטנדרטית, אך ניתן להוסיפם לארגז הכלים האישי.

הוספת פקדים לארגז הכלים נעשית בלחיצה ימנית על מקום כלשהו בארגז הכלים. בתפריט שנקבל נבחר באפשרות **Components** אשר תפתח חלון דו-שיח כדוגמת זה שמוצג בתרשים 3.11.

בחלון זה מוצגת רשימת פקדים. סימון ☑ משמאל לשם הפקד ולחיצה על Apply או OK גורמים לבחירה בו ולהצבתו בארגז הכלים; ביטול הסימון מסיר את הפקד מהארגז.

רשימה זו מייצגת קבצי OCX, הנמצאים בתיקיה System שבתיקיית מערכת ההפעלה. ניתן באופן "ידני" לדלות אחד מהקבצים האלה ולהציג את הפקד שאותו הוא מייצג לרשימה האמורה. פעולה זו מתבצעת באמצעות הלחצן Browse, אשר פותח תיבת דו-שיח המציגה ומאפשרת בחירת קבצי OCX.



תרשים 3.11

לא נדון במסגרת סעיף זה במה שמייצג כל שם ברשימת הפקדים. הנקודה החשובה והעיקרית היא שניתן לייבא קובץ OCX למערכת, למקם אותו בזיכרון ולהוסיף פקד לארגז הכלים.

תרגיל לסיכום הנושאים העיקריים שנלמדו בפרק

לצורך התרגיל נבנה את הטופס המוצג בתרשים 3.12.

תרשים 3.12

הטופס כולל את סוגי הפקדים האלה: ComboBox, ListBox, Label, HScrollBar, CommandButton ו-TextBox. מטרת התרגיל להרכיב משפט אשר יוצג ב-TextBox כאשר נלחץ על CommandButton. חלקי המשפט נתונים במספר תיבות שונות כפי שנראה בהמשך, והמשתמש ירכיב מהם את המשפט הרצוי, שמוצג בתחתית תרשים 3.13. בסיום התרגיל ובזמן ריצה הטופס צריך להיות דומה לזה שמוצג בתרשים.

תרשים 3.13

התיבה המשולבת (ComboBox) תציג לבחירה את חלקי המשפט: I am, He is, She is. תיבת הרשימה (ListBox) תציג לבחירה את חלקי המשפט: reading ו- learning. תפקיד פס הגלילה (HScrollBar) הוא, לאפשר בחירה של מספר בטווח 1-5 המייצג את מספר השעות (hours) במשפט. לצורך הרכבת המשפט "I am learning Visual Basic 2 hours every day" נבחר את חלק המשפט "I am" מהתיבה המשולבת. נבחר בפריט השני בתיבת הרשימה ונקבל את המילה "learning". מספר שעות הלימוד יקבעו באמצעות סרגל הגלילה. את שאר חלקי המשפט נשרשר בשורות הקוד. כאשר נלחץ על הלחצן Show יופיע המשפט השלם בתיבת הטקסט.

את התרגיל נבנה יחד בשלבים הבאים:

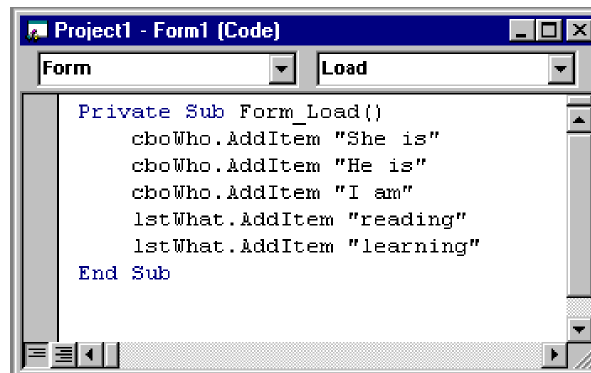
1. ניצור את הטופס כדוגמת תרשים 3.12.
2. נקבע לכל פקד את המאפיינים על פי טבלה 3.2.

טבלה 3.2

פקד	מאפיין	ערך
ComboBox	Name	cboWho
ListBox	Name	lstWhat
CommandButton	Name	cmdShow
	Caption	"Show"
HScrollBar	Name	hsbHours
	Min	1
	Max	5
Label	Name	lblHours
	Caption	(Empty)
TextBox	Name	txtSentence
	Text	(Empty)

3. בתיבות ComboBox ו-ListBox נמלא את הפריטים שהמשתמש יוכל לבחור מתוכם.

הוספת הפריטים לתוך התיבות תתבצע בזמן שהטופס נטען מהזיכרון. האירוע המתאים לדרישה זו הוא Form_Load( ). נעבור למבט קוד ונכתוב בשיגרה זו את שורות הקוד המודגמות בתרשים 3.14.



תרשים 3.14

4. תפקיד הפקד lblHours בטופס הוא להציג את הערך הנוכחי של פס הגלילה. הקישור בין השניים ייעשה בעזרת שורות קוד. האירוע המתאים הוא Scroll של הפקד hsbHours. שורת הקוד תיראה כך:

```
Private Sub hsbHours_Scroll()
      lblHours.Caption = hsbHours.Value
End Sub
```

5. לסיום, נותר לנו לדאוג לכך שכל חלקי המשפט יוצגו בסדר הנכון בתיבת הטקסט כאשר נלחץ על לחצן Show. האירוע המתאים הוא Click של הפקד cmdShow ושורות הקוד המתאימות תיראנה כך:

```
Private Sub cmdShow_Click()
      txtSentence.Text = cboWho & " " & _
      lstWhat & " Visual Basic " & _
      lblHours.Caption & " hours every day."
End Sub
```

בעזרת האופרטור "&" שרשרנו את כל חלקי המשפט לתוך תיבת הטקסט. כתיבת כל הפקודה בשורה אחת אינה מומלצת, ולכן השתמשנו בקיטוע המשפט על ידי קו-תחתון (_) המאפשר להמשיך את המשפט בשורה חדשה.

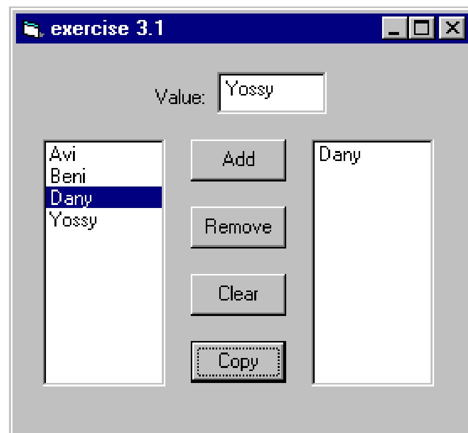
החלק הראשון במשפט הוא חלק המשפט שנבחר מתוך התיבה המשולבת. הפריט שנבחר מיוצג על ידי שם הפקד, במקרה שלנו - cboWho. שימוש בשם הפקד מקביל לשימוש בתכונה Text; במילים אחרות, כתיבת cboWho כמוה ככתיבת cboWho.Text.

באופן דומה שורשר הפריט (או חלק המשפט) מתוך תיבת הרשימה lstWhat. ביניהם שרשרנו רווח, כדי להפריד בין מילות המשפט. אחריהם באות המילים הקבועות שביניהן נמצא ערך השעות המתקבל מהתכונה Caption של הפקד lblHours.

6. כל שנותר, הוא להריץ את התוכנית (הקשת F5) ולהרכיב את המשפט הרצוי.

תרגול

1. צור את הטופס כדוגמת תרשים 3.15 :



תרשים 3.15

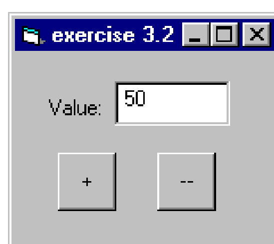
הקלדת מילה בתיבה **Value** ולחיצה על לחצן **Add** תגרום להוספת המילה שהוקלדה אל תיבת הרשימה שבצד שמאל.

בחירת אחד מהפריטים שבתיבת הרשימה השמאלית (על ידי לחיצה עליו) ולחיצה על לחצן **Remove**, תגרום למחיקת הפריט מהרשימה.

לחיצה על לחצן **Clear** תגרום למחיקת כל הפריטים שבתיבת הרשימה השמאלית.

בחירה באחד מהפריטים שבתיבת הרשימה השמאלית (על ידי לחיצה עליו) ולחיצה על לחצן **Copy**, תגרום להעתקת הפריט שנבחר מהרשימה השמאלית לתיבת הרשימה הימנית.

2. צור את הטופס כדוגמת תרשים 3.16 :



תרשים 3.16

אתחל את תיבת הטקסט **Value** בערך 50.

בלחיצה על לחצן + יגדל המספר שבתיבת הטקסט ב- 1.

בלחיצה על לחצן **-** יקטן המספר שבתיבת הטקסט ב- 1.

4: מאפיינים (Properties)

אחת הבעיות העיקריות של מתכנתי ויז'ואל בייסיק היא רצונם להפיק את המירב מהתוכנה על תכונותיה ויכולותיה הרבות. רבים אינם יודעים כיצד להפיק את המירב מהתוכנה, אלא לאחר עבודה והתנסות בתכנות במשך זמן רב. לעיתים יש תכנון נכון וטוב של היישום, והעיצוב נעשה גם הוא לאור חשיבה תכנותית יוצרת, האלגוריתמים לפתרון בעיות יחודיות גם הם ידועים והנה - התרגום של כל אלה לשפת ויז'ואל בייסיק אינו תקין. חלק מהיכולת לנצל את כישורי השפה מתבטא בהכרה טובה של המאפיינים של כל פקד ותפקידם. בפרק הקודם הכרנו את הפקדים ולמדנו להכיר חלק מהמאפיינים המיוחדים לכל אחד מהם. עתה נלמד את המאפיינים העיקריים והשימושיים ביותר, המשותפים לכלל הפקדים ואשר חוזרים על עצמם ברוב (חלקם אף בכל) היישומים שנפתח.

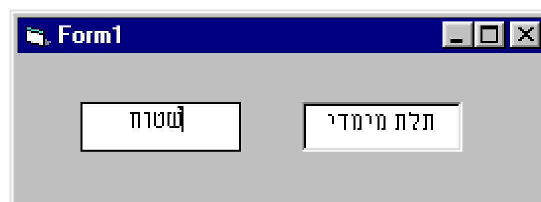
הכרת המאפיינים

Alignment - תכונה זו קובעת את כיוון יישור הטקסט הכתוב בפקד: **0** - יישור לשמאל, **1** - יישור לימין, **2** - למרכז (ראה תרשים 4.1).



תרשים 4.1

Appearance - קובע את סוג הופעת הפקד על גבי הטופס: **0** - קובע שהפקד יופיע באופן שטוח על גבי הטופס; **1** - מציג את הפקד בצורה תלת-מימדית (ראה תרשים 4.2).



תרשים 4.2

BackColor - צבע רקע של הפקד. ניתן לבחור אחד מצבעי המערכת (מתוך הכרטיסיה System), או את אחד הצבעים בלוח הצבעים (הכרטיסיה Palette).

BorderStyle - סוג מסגרת הפקד: **0** - ללא מסגרת, **1** - מסגרת שקועה (תלת-מימדית) (ראה תרשים 4.3).



תרשים 4.3

Caption - כיתוב. זהו הכיתוב שיופיע על פני הפקד (תרשים 4.4).

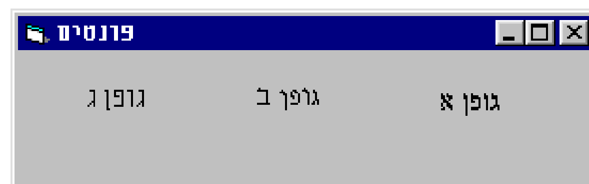


תרשים 4.4

ControlBox - תכונה זו שייכת לטופס (Form). לכל טופס יש בצידו הימני העליון שלושה לחצני בקרה, השולטים בסגירה, בשינוי גודל ובמזעור, בהתאמה. הערך **True** מורה ששלושת הלחצנים האלה יופיעו בטופס. השמת ערך **False** במאפיין זה תגרום להסתרתם.

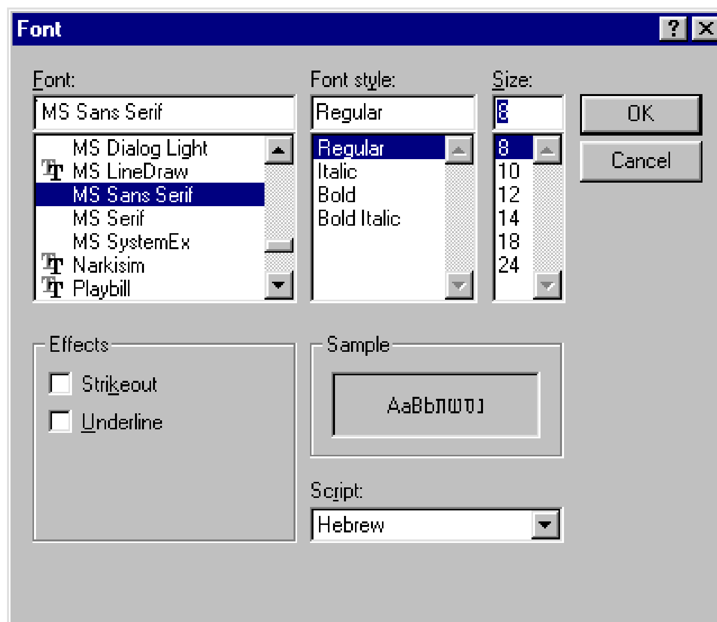
Enabled - תכונה זו קובעת את יכולת המשתמש להשתמש באובייקט או לא: **True** (ברירת המחדל) - האובייקט ניתן לשימוש; **False** - אין אפשרות גישה ושימוש באובייקט.

Font - קביעת סוג הגופן, הסגנון וגודל הטקסט המופיע בפקד (תרשים 4.5).



תרשים 4.5

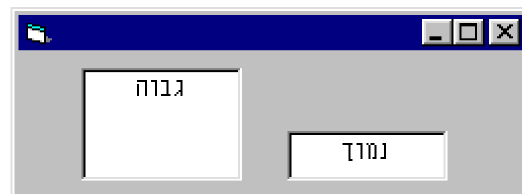
באמצעות מאפיין זה ניתן לקבוע גם את גודל הגופן, סגנונו ואפקטים נוספים. השינויים מתבצעים בתיבת הדו-שיח **Font** שמוצגת בתרשים הבא:



תרשים 4.6

ForeColor - צבע חזית.

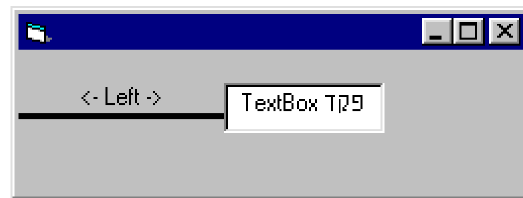
Height - גובה האובייקט (תרשים 4.7).



תרשים 4.7

Index - אם האובייקט הוא חלק ממערך, תכונה זו קובעת את מיקומו, או במילים אחרות, את מספר האינדקס שלו במערך.

Left - המרחק בין הגבול השמאלי של הטופס לבין הגבול השמאלי של הפקד (תרשים 4.8).



תרשים 4.8

MousePointer - קובע את צורת העכבר כאשר הוא נע על פני האובייקט.

RightToLeft - תכונה זו שימושית מאוד ליישומים הכתובים למשתמש העברי. כשמציבים לתכונה זו את הערך **True**, הפקד הופך כיוון לימין-שמאל (כמקובל בעברית). הדבר מתבטא בכיוון כתיבה מימין לשמאל, כותרות ותפריטים בצד ימין של הטופס וכו'.

TabIndex - תכונה זו קובעת את סדר התנועה של המשתמש בטופס. התנועה בטופס והמעבר מפקד אחד למשנהו נעשית באמצעות המקש **Tab**. הסדר הרצוי נקבע בעזרת תכונה זו: ערך **0** מורה כי הפקד יקבל ראשון את המיקוד, הפקד הבא אחריו יקבל ערך **1**, וכן הלאה לפי סדר עולה.

TabStop - תכונה זו קובעת אם הפקד יקבל מיקוד (לחיצת המשתמש על מקש Tab) או לא. הערך **True** (ברירת המחדל) מאפשר קבלת מיקוד, והערך **False** גורם לדילוג על הפקד והתמקדות בפקד הבא אחריו בסדר.

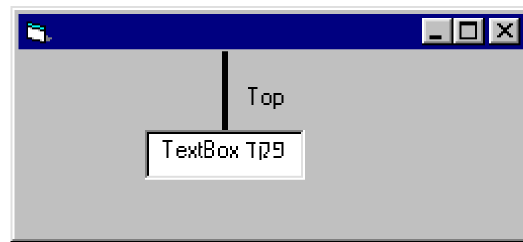
בניגוד למאפיין **Enabled**, שבו ערך False קובע כי אין אפשרות להשתמש בפקד, במאפיין TabStop הערך False גורם לדילוג על הפקד. עם זאת, בידי המשתמש "לשבור" את הרצף ולתת מיקוד לפקד ולהשתמש בו.

ToolTipText - מוכרת לנו התופעה השכיחה בסביבה חלונאית, בה מניחים את העכבר על פקד מסוים ולאחר שנייה או שתיים מופיעה תיבה, המסבירה לנו מהו תפקיד הפקד. התכונה ToolTipText קובעת את **טקסט תיאור הכלי** אשר מוצג באותה תיבה המתקבלת בעת השהיית העכבר על הפקד (תרשים 4.9).



תרשים 4.9

Top - המרחק בין הגבול העליון של הטופס לבין הגבול העליון של הפקד (תרשים 4.10).



תרשים 4.10

Visible - תכונה זו קובעת אם פקד יוצג בזמן הפעלת התוכנית, או לא: הערך **True** (ברירת המחדל) מציג את הפקד בזמן ריצה; **False** גורם להסתרת הפקד.

במאפיין זה נשתמש כשנרצה להחזיק ערך כלשהו בפקד, או להציב אובייקט עזר על פני הטופס, אשר יהיה נוח לשימוש לנו כמתכנתים, ולא נרצה שהוא ייראה בזמן ריצה.

והנה טיפ קטן, אבל חשוב. במהלך הפעלת התוכנית נרצה לפעמים להעביר פרמטר מטופס אחד למשנהו. בשפת ויז'ואל בייסיק אין כלי או פונקציה מיוחדים לצורך זה. אולם, עם טיפת תושיה ובעזרת המאפיין Visible נתגבר על הבעיה. בטופס ה"קורא" (הטופס שממנו נרצה להעביר את הארגומנט לטופס האחר שנפתח) ניצור Label או TextBox אשר יאותחלו בערך אשר נרצה להעביר כפרמטר, ואת ערך תכונת Visible שלו נגדיר False. לאחר מכן, כשנטען את הטופס ה"נקרא" (הטופס החדש שנפתח ואשר צריך לקבל את הפרמטר שנשלח), נקרא מתוכו את תוכן הפקד המוסתר שבטופס ה"קורא" אשר מחזיק בערך הפרמטר. נניח ששם הטופס הקורא הוא frmMain והאובייקט המחזיק את ערך הפרמטר הוא txtParameter (מסוג TextBox).
הפנייה מהטופס ה"נקרא" אל האובייקט תיעשה באמצעות שורת הקוד הזו:

```
frmMain.txtParameter.Text
```

Width - רוחב האובייקט.

5: משתנים

בכבל שפת תכנות, גם בוויז'ואל בייסיק נמצא את אבני הבניין של השפה - **המשתנים**.

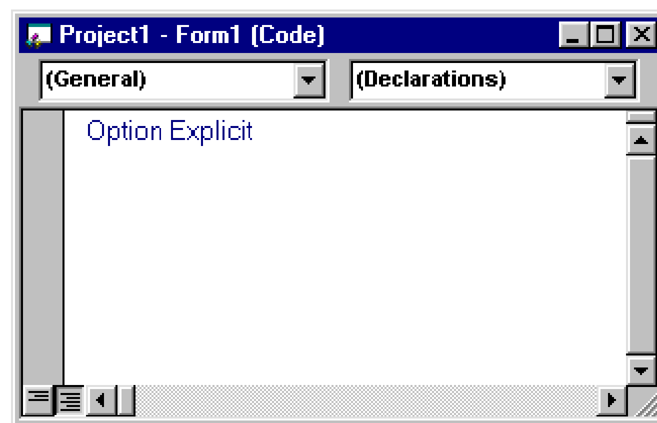
ברוב השפות שהכרנו עד היום (C,C++, Pascal וכו'), הצהרת המשתנה היתה חייבת להקדים את השימוש בהם. שפת ויז'ואל בייסיק שונה משפות אלו בכך שבה השימוש במשתנה אינו מחייב הצהרה קודמת שלו. משפט זה,

```
MyName = "Dany"
```

הוא משפט חוקי ותקף, למרות שלא הצהרנו מראש על המשתנה MyName. עם זאת, מומלץ להצהיר תמיד על משתנים, לשם תיעוד וניהול טוב יותר שלהם, כפי שנלמד בהמשך.

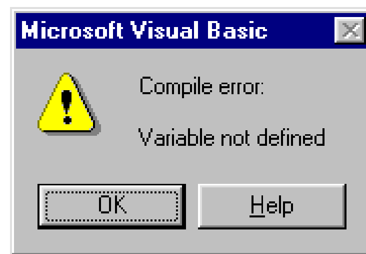
המשפט Option Explicit

ההצהרה "Option Explicit" שנרשום בחלון הקוד בחלק General Declarations שבראש הטופס (ראה תרשים 5.1), מורה על כך שאנו דורשים שקודם השימוש במשתנה כלשהו בתוכנית, עלינו להצהיר עליו. בדרך זו אנו כופים הצהרה על המשתנה לפני השימוש בו.



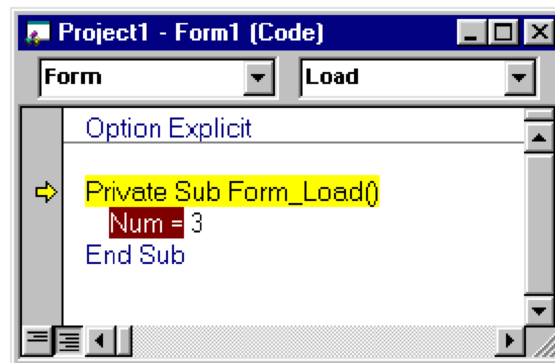
תרשים 5.1

הצהרה זו גורמת לוויז'ואל בייסיק להתריע בכל פעם שהיא מוצאת משתנה שלא הוצהר במפורש קודם לשימוש בו. ההתרעה מופיעה כהודעת שגיאת הידור (תרשים 5.2).



תרשים 5.2

אישור ההודעה גורם לוויז׳ואל בייסיק לסמן את המשתנה בתוכנית שלא הוצהר קודם כתיבתו (תרשים 5.3). בנוסף לסימון המשתנה מסומנת גם השיגרה או הפונקציה שבהן נעשה שימוש במשתנה הלא מוצהר.



תרשים 5.3

הצהרה זו שימושית במיוחד כשאנו חוששים משגיאות הקלדה, למשל. במקרה זה, ההצהרה Option Explicit תסייע לאתר שגיאות מסוג זה.

המשתנה Variant

כפי שראינו, אין צורך להצהיר על משתנים, כשאין כותבים את ההצהרה המיוחדת Option Explicit. עם זאת, כאשר ויז׳ואל בייסיק פוגשת במשתנה שלא הצהרנו קודם לכן על סוגו, היא מגדירה אותו באופן אוטומטי כמשתנה מסוג Variant. משתנה זה מסוגל להחזיק נתונים בכל גודל ובכל תבנית, זהו משתנה ״כללי״. הרווחנו אמנם את הצורך בהצהרה, אך גם בזבזנו הרבה מקום בזיכרון. לכן מומלץ להצהיר מראש על סוגי המשתנים על פי הייעוד הצפוי בתוכנית. באופן זה, כל משתנה תופס את המקום בזיכרון שהוא זקוק לו.

סוגי משתנים

ויז'ואל בייסיק מכילה סוגים רבים של משתנים. בטבלה הבאה נציג את המשתנים העיקריים והשימושיים ונפרט את הסוג, המשמעות, הגודל בבתים ואת טווח הערכים.

טבלה 5.1

סוג המשתנה	**משמעות**	**גודל (בבתים)**	**טווח הערכים**
Byte	בית אחד	1	0 - 225
Integer	שלם	2	-32,768 - 32,767
Long	שלם ארוך	4	-2,147,483,648 - 2,147,483,647
Single	נקודה צפה	4	-3.402823E38 - 3.402823E38
Double	כפול	8	-1.79769313486232E308 - 1.79769313486232E308
String	מחרוזת	8	מ- 0 עד 2 מיליארד (ביליון) תווים
Currency	מטבע	1 לכל תו	-922,337,203,685,477.5808 - 922,337,203,685,477.5807
Date	תאריך	8	
Boolean	בוליאני (אמת/שקר)	2	True or False
Variant	כללי	24	בהתאם לסוג המשתנה

משתנים מספריים

בוויז'ואל בייסיק קיימים מספר סוגי משתנים מספריים: Single ,Long ,Integer, Double ו- Currency. כל סוג משתנה "תופס" מקום שונה בזיכרון על פי גודלו המוגדר. באותה מידה, טווח הערכים של כל משתנה מותאם לגודלו. מטרת החלוקה למספר רב של סוגים היא בעיקר החיסכון בזיכרון. אם לדוגמה, נשתמש בלולאת For...Next במהלך התוכנית ונצהיר על משתנה מונה הלולאה מסוג Single למשל, אנו שוגים. ברור לנו שלמטרה זו דרוש משתנה שמייצג מספר שלם בלבד וערכו נמוך בדרך כלל. לכן, נעדיף במקרה זה להצהיר על משתנה הלולאה כמשתנה מסוג Integer ובכך גם נחסוך 2 בתים (ההפרש בין Single ל- Integer) בזיכרון. ולאלה שפוקחים עיניים ושואלים "אז מה אם גזלת עוד שני בתים?", ברצוני לומר שבתוכנית יש לולאות רבות, ויש עוד מקומות רבים שאפשר לחסוך בהם.

משתנה מחרוזת

כאשר נזדקק למשתנה אשר יכיל מחרוזת, נצהיר עליו כעל משתנה מסוג String. באופן רגיל, גודל המשתנה מותאם למחרוזת שהוא מכיל. כך, אם במהלך התוכנית המחרוזת קטנה - המשתנה יקטן, ואם היא גדלה - המשתנה יתפוס יותר בתים. עם זאת, ניתן להגדיר גודל **קבוע** למשתנה על ידי ציון הגודל הרצוי. לדוגמה,

```
Dim strName As String * 2
```

במקרה זה המשתנה strName יכיל שני תווים בלבד. אם נרשום את שורות הקוד האלו,

```
strName = "Dany"
Print strName
```

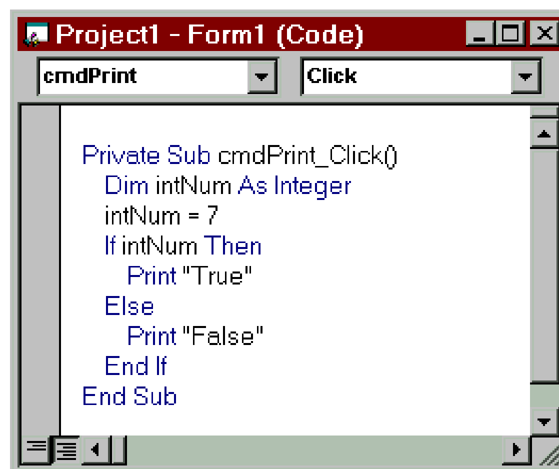
התוצאה תהיה, שעל המסך יודפס "Da", ולא "Dany".

משתנה בוליאני

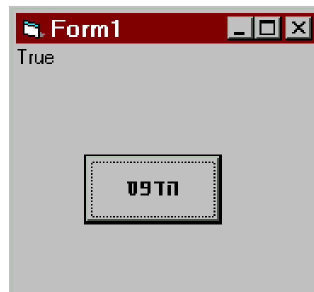
כאשר דרוש משתנה אשר יכיל מידע של שני מצבים, כמו אמת/שקר (True/False), כן/לא (Yes/no) או משתנה "דגל" (Flag) המכיל ערך דלוק/כבוי (On/Off), נצהיר עליו כעל משתנה מסוג Boolean. ככלל, הערכים שמקבל משתנה בוליאני הם True/False, וברירת המחדל היא False.

ניתן להשתמש במשתנה מספרי אשר יתנהג כמשתנה בוליאני. הערכים המוגדרים בו הם ערכים מספריים רגילים (ולא True/False), אולם מוסכם שערך **0** (אפס) מציין ערך שקרי (False) וכל שאר הערכים (שאינם אפס) מציינים ערך אמת (True).

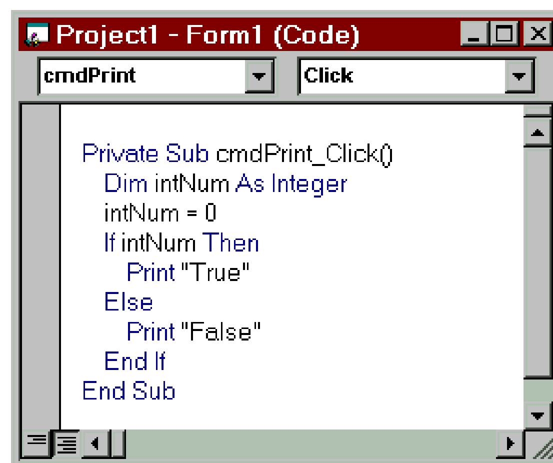
תרשימים 5.4 - 5.7 מדגימים שימוש במשתנה מספרי המתנהג כמשתנה בוליאני.



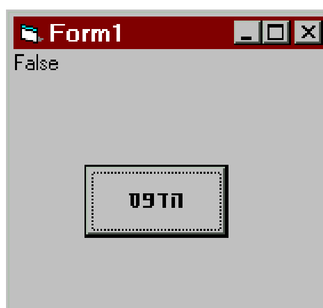
תרשים 5.4



תרשים 5.5: תרשים 5.4 בזמן ריצה



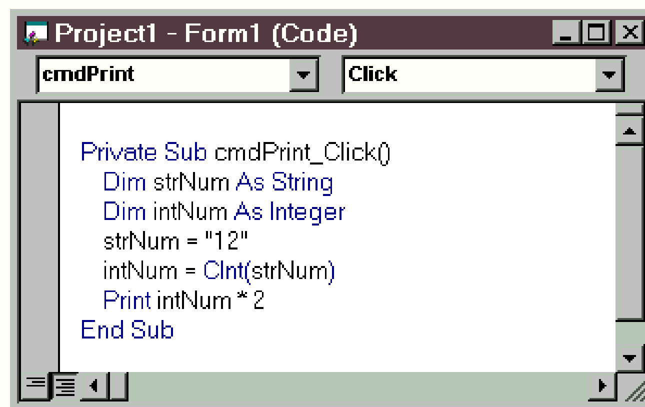
תרשים 5.6



תרשים 5.7: תרשים 5.6 בזמן ריצה

המרת סוגי משתנים

בוויזואל בייסיק יש פונקציות הממירות ערכים מסוג מסוים של משתנים לסוג אחר. אם לדוגמה, משתנה מחרוזת מכיל את הערך "123", אשר יכול לייצג עבורנו מספר, אין אפשרות לבצע על המשתנה חישובים מתמטיים כלשהם כמו על מספר. עם זאת, ניתן להצהיר על משתנה מספרי מתאים, ולהמיר אליו את המחרוזת (תרשים 5.8). מעתה יהיה ניתן לבצע חישובים במשתנה המספרי (תרשים 5.9).



תרשים 5.8



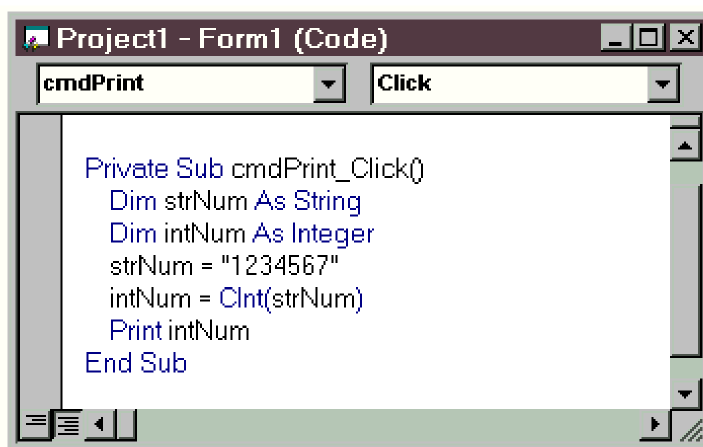
תרשים 5.9

טבלה 5.2 מפרטת את פונקציות ההמרה האפשריות. ניתן להמיר כל משתנה לכל משתנה.

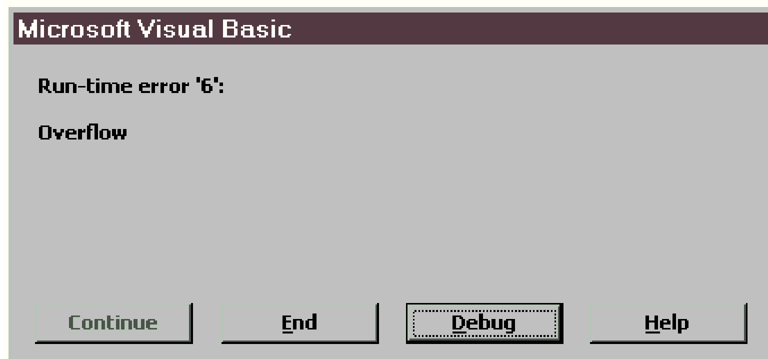
טבלה 5.2

הפונקציה	ממירה ל-
CBool()	Boolean
CByte()	Byte
CCur()	Currency
CDate()	Date
CDbl()	Double
CInt()	Integer
CLong()	Long
CSng()	Single
CStr()	String
CVar()	Variant
CVErr()	Error

יש לשים לב שכאשר ממירים משתנה מסוג אחד לסוג משתנה אחר, המשתנה חייב להיות חוקי לסוג המשתנה החדש. לדוגמה, אין להמיר את משתנה המחרוזת "1234567" למשתנה מסוג Integer (תרשים 5.10), מכיון שהמספר 1,234,567 אינו בטווח המספרים שמכיל סוג המשתנה Integer. במקרה כזה נקבל הודעת שגיאה (תרשים 5.11) וההמרה לא תבוצע. כפתרון יש להמיר לסוג משתנה Long.



תרשים 5.10



תרשים 5.11

ערכים מיוחדים

ערכים אלה אינם משתנים אלא ערכים הקיימים (built in) בוויז׳ואל בייסיק, היכולים לקבל סוגים שונים של משתנים.

Empty – ריק

הערך Empty מצביע על כך שמשתנה מסוג Variant הוא ריק. במילים אחרות, עדיין לא בוצעה במשתנה פעולת השמה כלשהי מרגע שהוצהר. ברגע שמתבצעת פעולת השמה והמשתנה מקבל ערך חדש (ובכלל זה הערכים: 0, "" ו- Null), "נעלם" הערך Empty. ניתן לאתחל שוב את המשתנה בערך זה על ידי הצבת ערך Empty למשתנה בפעולת השמה רגילה. לדוגמה:

```
MyVariant = Empty
```

הערך Empty **אינו** ערך **0** (אפס) או מחרוזת ריקה (""), אלא הוא ערך בפני עצמו. ניתן לבחון משתנה כלשהו אם הוא מכיל ערך Empty או לא, באמצעות הפונקציה IsEmpty() המחזירה True כאשר המשתנה מכיל ערך זה.

כאשר משתנה המכיל ערך Empty הוא חלק מביטוי, ויז׳ואל בייסיק מתייחסת אליו כאל 0 (אפס), או מחרוזת ריקה (""), בהתאם לסוג המשתנה.

Null – "אין"

Null הנו ערך מיוחד שיכול לקבל משתנה מסוג Variant. ערך זה שימושי בעיקר במסדי נתונים ומורה על "חוסר" ערך במשתנה.

ניתן לאתחל משתנה מסוג Variant בערך Null בפקודת השמה רגילה, בעזרת מילת המפתח Null, לדוגמה:

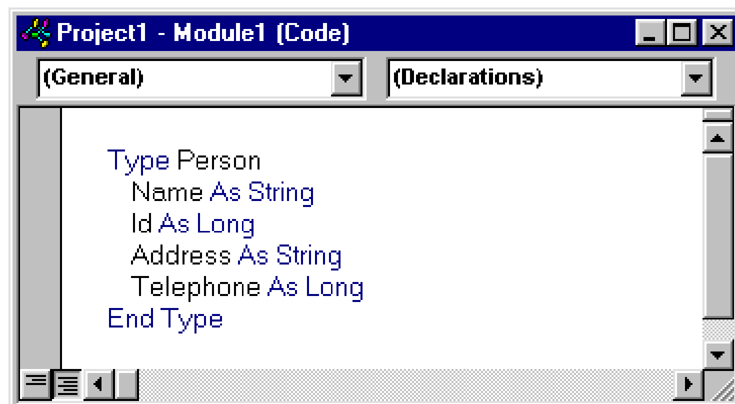
```
MyData =  Null
```

כדי לבדוק אם משתנה מכיל ערך Null השתמש בפונקציה ()IsNull, לדוגמה:

```
If IsNull(MyData) Then
      MsgBox "MyData is Null!"
End If
```

מבנים המוגדרים על ידי המשתמש (User-Defined Type)

מבנה (Struct) מוכר לנו משפות תכנות שונות כ- C ו- ++C. המבנה הוא משתנה שמכיל בתוכו קבוצת משתנים מסוגים שונים. השימוש במבנה נעשה בעיקר כאשר לקבוצת המשתנים השונים יש מכנה משותף. לדוגמה, מחלקת משאבי אנוש בחברה מעוניינת להחזיק פרטים אישיים של כל עובדיה לצורך ניהול וחישוב משכורתם. במקום להצהיר על משתנים מסוגים שונים עבור שם העובד, כתובתו וכו', נוח וברור יותר להחזיק משתנה אחד מסוג מבנה, אשר יכיל בתוכו את כל סוגי המשתנים השונים המרכיבים יחד את פרטי העובד. משתנה מסוג **מבנה** מוצהר בתוכנית באמצעות המילה Type. הצהרה על משתנה-מבנה מוצגת בתרשים 5.12.



תרשים 5.12

הבה נסביר את המבנה המוצהר כאן:

Person הוא שם המשתנה - המבנה, המכיל את סוגי המשתנים השונים הכוללים את פרטי העובד. כאשר הצהרנו על המבנה **יצרנו** סוג חדש של משתנה. מעתה ניתן להצהיר על משתנה חדש מסוג המבנה Person.

לדוגמה,

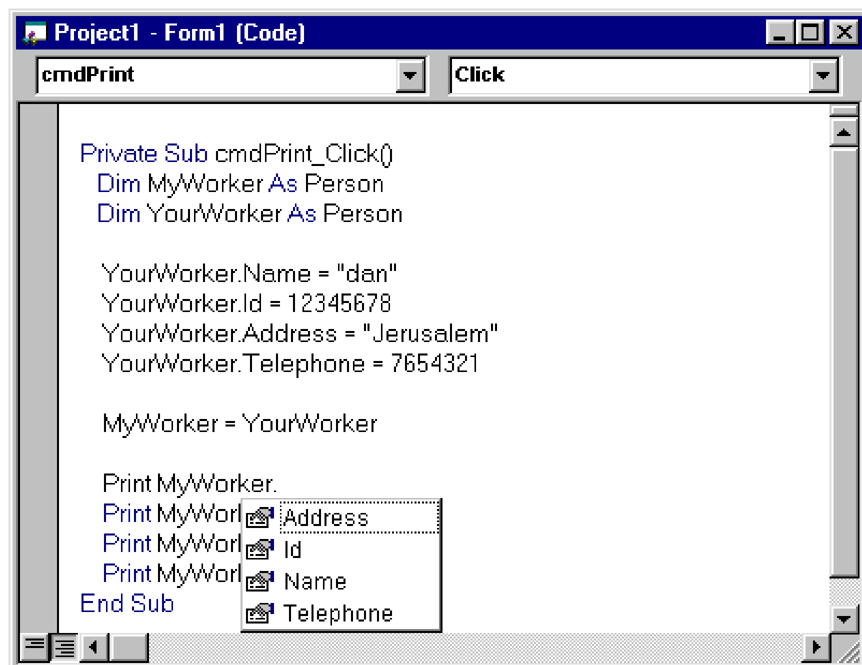
```
Dim Employee As Person
```

Employee הוא עתה משתנה מסוג המבנה Person וגם לו ארבעה משתנים שונים שהוא יכול לפנות אליהם.

כשרוצים לפנות למשתנה מסוים לחישוב או לפעולת השמה, צריך לרשום את שמו. במקרה של משתנה שהוא חלק מקבוצת משתנים בתוך מבנה, הדבר שונה. כדי לפנות למשתנה כזה, יש לפנות בראשונה אל משתנה המבנה שהוא שייך לו, ורק אחר כך לפנות אליו. אם נניח, לצורך הדוגמה, כי יש לנו משתנה Employee מסוג Person המכיל נתוני עובד, וברצוננו לעדכן את שם העובד ל-Dan, נעשה זאת כך:

```
Employee.Name = "Dan"
```

תחילה פונים אל משתנה המבנה Employee, ורק אחר כך אל אחד המשתנים שכלולים בו, ובמקרה זה - Name. המבנה הוא משתנה חדש שיצרנו במערכת, אך ויז'ואל בייסיק מזהה אותו ומכירה את כל המשתנים שהוא כולל. כאשר נרשום את שם המשתנה (המבנה) ולאחריו נקודה, מייד תוצג תיבת רשימה המכילה את כל משתני המבנה (תרשים 5.13) כדי שנוכל לבחור מתוכה את המשתנה הרצוי.

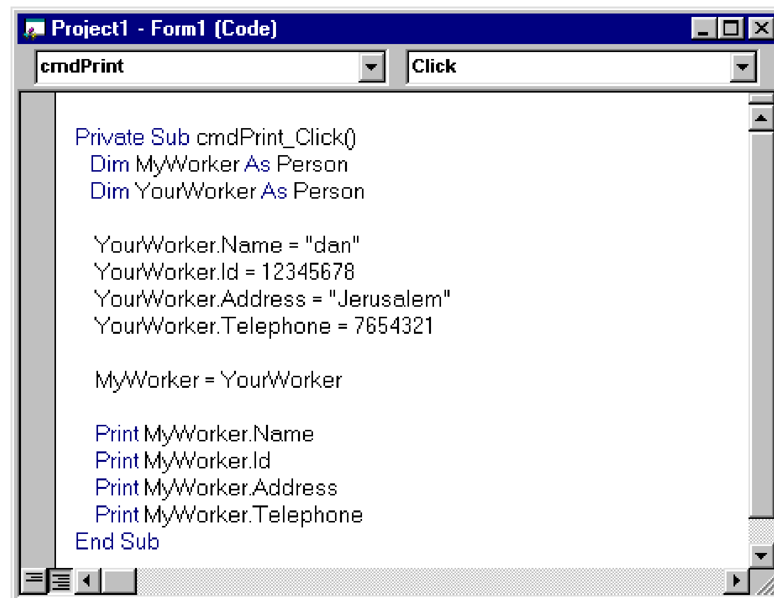


תרשים 5.13

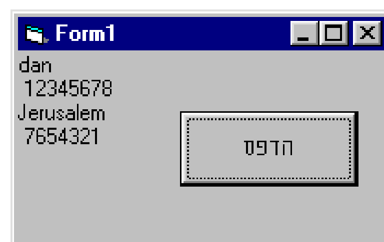
באופן דומה ניתן להעתיק משתנה-מבנה אחד אל משתנה-מבנה אחר, על כל סוגי משתניו הפנימיים. חשוב לציין, שסוג המבנה של שני המשתנים חייב להיות זהה! לדוגמה,

```
Dim MyWorker As Person
Dim YourWorker As Person
MyWorker = YourWorker
```

במקרה זה ערכי **כל** המשתנים ב- YourWorker יושמו לתוך המשתנים ב- MyWorker (תרשים 5.14-5.15).



תרשים 5.14



תרשים 5.15

הצהרה על משתנים

אופן ההצהרה נעשה באמצעות המשפט **Dim**. ההצהרה כוללת את שם המשתנה ואת סוגו. ככלל, הצהרה על משתנה נעשית כך :

```
Dim Variable As VariableType
```

לדוגמה:

```
Dim Name As String
Dim Id As Long
Dim Num As Integer
Dim Birth As Date
Dim Bool As Boolean
```

המילים Dim, As וסוג המשתנה (לדוגמה, String) הן מילים שמורות ונצבעות בכחול באופן אוטומטי.

ניתן להצהיר במשפט Dim אחד על כמה משתנים מאותו סוג. לדוגמה,

```
Dim Name As String, Address As String
```

שים לב להצהרה הזו:

```
Dim Name, Address As String
```

הצהרה זו **אינה** זהה לקודמתה! במקרה זה, ויז׳ואל בייסיק תזהה את המשתנה Address בלבד כמשתנה מסוג String. המשתנה Name, במקרה זה, אינו מסווג וההתייחסות אליו תהיה כאל משתנה Variant.

באופן דומה, ניתן לנצל את ההצהרה Dim פעם אחת עבור סוגים שונים של משתנים. לדוגמה,

```
Dim Num As Integer, Birth As Date, Bool As Boolean
```

שמות משתנים

כשקובעים שם למשתנה, רצוי לקבוע לו שם אשר מותאם לערך שהוא מכיל. לדוגמה, למשתנה שיכיל ״שם פרטי״ נקרא FirstName. כמו כן, מקובל לכתוב את שמות המשתנים באות רישית גדולה, אך אין זה חובה. לעומת זאת, יש חוקים למתן שם למשתנה אשר חייבים להתקיים, כדי ששם המשתנה יהיה תקף. החוקים לשם משתנה תקף הם:

1. שם המשתנה חייב להתחיל באות, אך יכול לכלול גם ספרות.
2. שם המשתנה אינו יכול לכלול שם השמור לסוג משתנה (לדוגמה String).
3. שם המשתנה לא יהיה ארוך מ- 255 תווים.
4. שם המשתנה לא יהיה כשם מילת מפתח (לדוגמה Loop).
5. ניתן לכלול בשם המשתנה קו תחתון (״_״) מלבד אותיות וספרות.

כללים **זהים** חלים גם על מתן שמות לפקדים באמצעות המאפיין **Name**. עם זאת, יש שני הבדלים בין מתן שמות למשתנים ובין מתן שמות לפקדים:

1. אורך שם הפקד (וכן הטופס, המודול והמחלקה) לא יעלה על 40 תווים.

2. ניתן להשתמש במילה שמורה כשם פקד אך ורק בצירוף סוגריים מרובעים.

לדוגמה,

```
[Loop].Caption = "Enter your name:"
```

במקרה זה שם הפקד "Loop" הינו שם חוקי. לעומת זאת, השורה הבאה תגרום להודעת שגיאה:

```
Loop.Caption = "Enter your name:"
```

השימוש בסוגריים מרובעים עבור שם שהוא מילה שמורה בוויז'ואל בייסיק הינו חוקי אך ורק בשם פקד **ואינו** חוקי בשם משתנה.

דוגמאות לשמות משתנים חוקיים:

```
FirstName
Address123
Id
Home_Telephone
```

דוגמאות לשמות משתנים חוקיים, אך **לא** מומלצים:

```
A
z
B1
B2
```

דוגמאות לשמות משתנים **לא** חוקיים:

המשתנה משתמש במילה שמורה:

```
Select
```

המשתנה אינו מתחיל באות:

```
9Year
```

המשתנה מכיל פסיק:

```
Last,Name
```

קידומת למשתנים

כמקובל במתן שמות לפקדים, גם בשמות משתנים נהוג לצרף לשם המשתנה קידומת המעידה על סוגו, כדי להקל על המתכנת והבודק את התוכנית. טבלה 5.3 מפרטת את סוגי המשתנים והקידומת המקובלת לכל אחד מהם:

טבלה 5.3

סוג משתנה	קידומת
Boolean	bln
Byte	byt
Currency	cur

קידומת	סוג משתנה
dtm	Date(Time)
dbl	Double
err	Error
int	Integer
lng	Long
sng	Single
str	String
udt	User-Defined Type (מבנה)
vnt	Variant

סוגי הצהרה

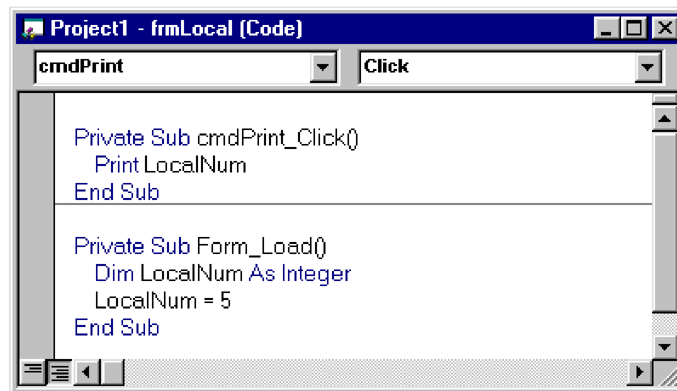
בשלושה מקומות בתוכנית ניתן להצהיר על משתנים, ומהם נגזרים שלושה סוגים שונים של הצהרה. לכל סוג משתנה יש טווח הכרה ואורך חיים שונה.

טווח הכרה (Scope) - הוא הטווח בו מוכר ערכו של המשתנה.

אורך חיים (Life) - הוא הזמן אשר המשתנה שומר על ערכו.

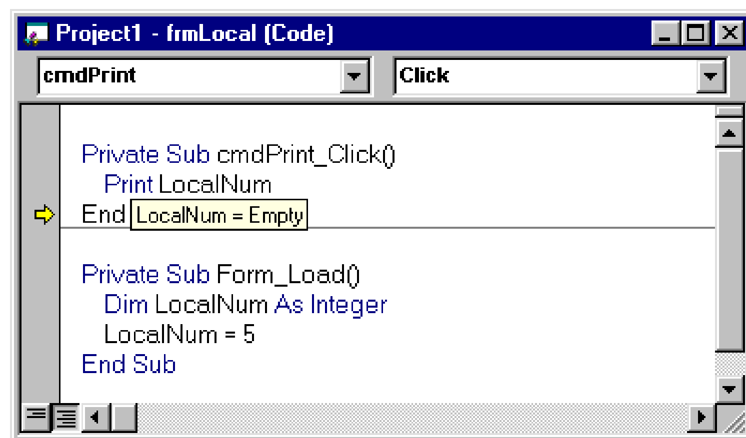
סוג ההצהרה הראשון הוא **ברמת האובייקט**. לכל אובייקט יש אירועים שונים או שגרות שהוא יודע להגיב להם. לצורך ביצוע פעולות מסוימות בשיגרה נזדקק לרוב גם למשתנים. משמעות ההצהרה ברמת האובייקט היא, שהמשתנה וערכו מוכרים באירוע או בשיגרה שבהם הוא הוצהר **בלבד**. אובייקט אחר, על אף שהוא באותו טופס לא יכיר את המשתנה הזה. מקובל לכנות משתנה מסוג זה בשם **משתנה לוקאלי**, או **משתנה מקומי**. במילים אחרות, טווח ההכרה של המשתנה הוא **רק** בשיגרה שבה הוצהר. בהתאם, גם אורך החיים של המשתנה הוא למשך האירוע **בלבד**. כאשר השיגרה או האירוע מסתיימים, מסתיים גם אורך החיים של המשתנה, והמקום בזיכרון אשר הוקצה לו משתחרר.

בתרשים 5.16 המשתנה LocalNum מוצהר כמשתנה מספר (Integer). המשתנה מוצהר בתוך השיגרה ()Form_Load ולכן הוא משתנה מקומי, ולפיכך, טווח ההכרה ואורך החיים שלו הם לשיגרה בה הוצהר בלבד. כאשר המשתנה מאותחל בשיגרה ומקבל את הערך 5, הוא שומר על ערכו רק בתחום השיגרה ()Form_Load.



תרשים 5.16

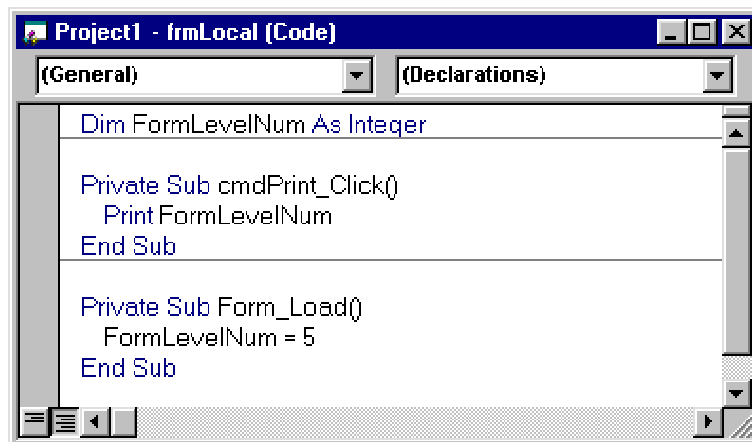
כאשר נשתמש במשתנה LocalNum בשיגרה אחרת (תרשים 5.17), משתנה זה כבר איבד את ערכו (במקרה זה, במקום 5 ערכו הוא Empty). המשתנה LocalNum בשיגרה ()cmdPrint_Click הוא משתנה **אחר** לגמרי. משתנה זה הוא מקומי בשיגרה שלו, ומכיון שלא אותחל, ערכו Empty. אורך החיים של המשתנה LocalNum בשיגרה Form_Laod מסתיים כאשר השיגרה מסיימת את פעולתה, והזיכרון משתחרר.



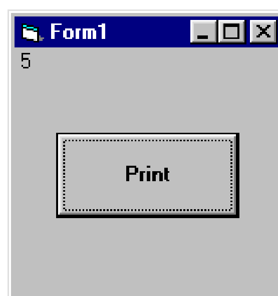
תרשים 5.17

סוג ההצהרה השני הוא **ברמת הטופס** (Form). בחלק העליון של הטופס (במבט קוד) מוקצה מקום להצהרות משתנים, General Declarations. הצהרה על משתנה באזור זה היא הצהרה ברמת הטופס. משתנה כזה וערכו יוכרו **לכלל** השגרות והפונקציות שנמצאות בטופס. במקרה זה אין צורך להצהיר בכל פעם מחדש בתוך כל שיגרה ופונקציה על המשתנה, כי הוא מוכר לכל.

בתרשים 5.18 מוצהר המשתנה FormLevelNum ברמת הטופס. משתנה זה מוכר לכל השגרות וכאשר נאתחל אותו בשיגרה Form_Load ונדפיס אותו בשיגרה ()cmdPrint_Click (תרשים 5.19) הוא עדיין ישמור על ערכו.



תרשים 5.18



תרשים 5.19

כאשר שיגרה כלשהי משנה את ערך המשתנה, ערכו החדש יוכר גם כאשר נשתמש בו בשיגרה אחרת. במילים אחרות, טווח ההכרה של המשתנה הוא **בכל** השגרות והפונקציות של הטופס בו הוצהר. אורך החיים של משתנה מסוג זה הוא כאורך חיין של הטופס, וכל עוד לא שחררנו את הטופס מהזיכרון המשתנה קיים ושומר על ערכו.

סוג ההצהרה השלישי הוא **ברמת המודול** (Module), קטע התוכנית. הצהרה על משתנה ברמת המודול מתחלקת לשניים: **Private** (פרטית) ו- **Public** (ציבורית).

משתנה המוצהר במודול כ- **Private**, מוכר לכל השגרות והפונקציות ש**בו** בלבד, ואין הוא מוכר לטפסים ומודולים אחרים. אופן ההצהרה נעשה כרגיל, אלא שהמילה Private מחליפה את מילת המפתח Dim.

לדוגמה:

```
Private FirstName As String
```

טווח ההכרה של משתנה זה הוא לכל השגרות והפונקציות של המודול, ואורך החיים שלו כאורך חיי המודול. תכונה זו תקפה בתנאי שהמשתנה הוצהר בחלק ההצהרות (General Declaration) של המודול. משתנה המוצהר בתוך אחת השגרות, או בתוך

הפונקציות במודול, הוא משתנה מקומי לאותה שיגרה או פונקציה. טווח ההכרה ואורך החיים שלו הם כשל משתנה מקומי רגיל.

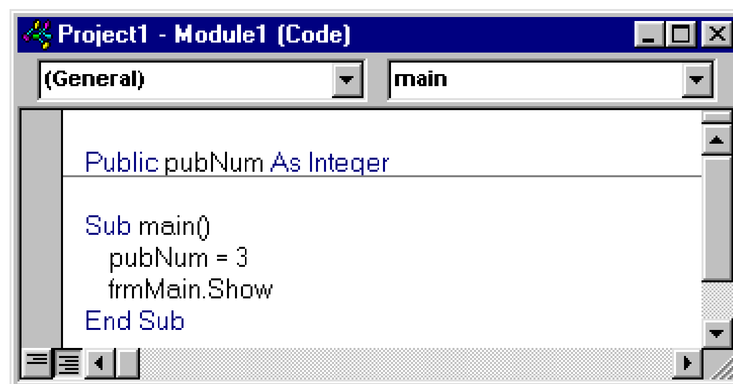
אפשרות שנייה היא להצהיר על המשתנה כ- **Public**.

לדוגמה:

```
Public FirstName As String
```

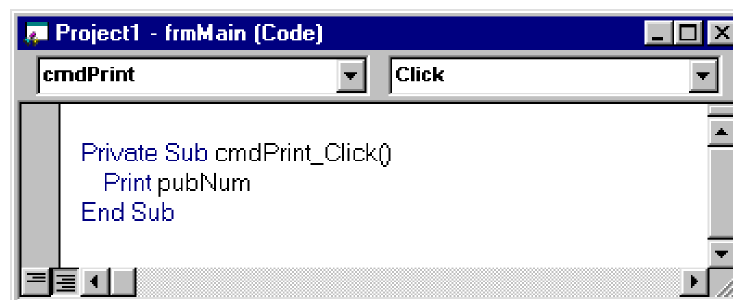
הצהרה על משתנה ברמת המודול כ- Public משמעותה שהמשתנה מוכר לא רק בכל השגרות והפונקציות של המודול בו הוא הוצהר, כי אם בכל המודולים והטפסים בפרוייקט. במילים אחרות, טווח ההכרה ואורך החיים של המשתנה הם למהלך כל התוכנית.

בתרשים 5.20 מוצהר המשתנה pubNum כמשתנה מסוג Public, והוא מאותחל בשיגרה Sub Main().

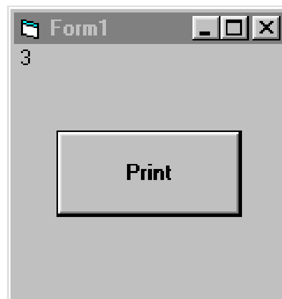


תרשים 5.20

המשתנה pubNum הוצהר כ- Public ומוכר בכל הטפסים והמודולים בפרויקט. מכיון שכך, כשנדפיס אותו מתוך השיגרה cmdPrint_Click() שבטופס frmMain (תרשים 5.21 - 5.22) הוא ישמור על ערכו.



תרשים 5.21



תרשים 5.22

ראוי לשים לב שמשתנה המוצהר כ- Public יהיה מוכר לכל המודולים והטפסים אך ורק אם הוא מוצהר במודול (Module). משתנה ברמת הטופס מוכר לפונקציות והשגרות של הטופס עצמו בלבד.

ניתן לציין בהצהרת המשתנה (ברמת המודול) את המילה השמורה Global במקום המילה השמורה Public. גם כך נקבל את אותה תוצאה: טווח הכרה ואורך חיים לכל אורך התוכנית.

משתנה סטטי

סוג משתנה נוסף הוא המשתנה הסטטי (Static variable). משתנה סטטי משלב בתוכו תכונות של משתנה מקומי ושל משתנה מודולרי. טווח ההכרה של משתנה סטטי הוא כזה של משתנה מקומי, ולכן הוא מוכר בפונקציה או בשיגרה שבה הוצהר בלבד. לעומת זאת, אורך החיים של משתנה זה הוא כזה של משתנה מודולרי, והוא אינו מאבד את ערכו בסיום הפונקציה או השיגרה. בכל פעם שנקרא לפונקציה או לשיגרה, ניווכח שהמשתנה שומר על ערכו הקודם. משתנה סטטי אינו משתחרר מהזיכרון בכל פעם שמסתיימת הפונקציה או השיגרה, ולכן הוא גם אינו "נולד" מחדש; זהו אותו משתנה שקיים במשך חיי המודול.

הצהרה על משתנה כסטטי נעשית בעזרת מילת המפתח Static. לדוגמה,

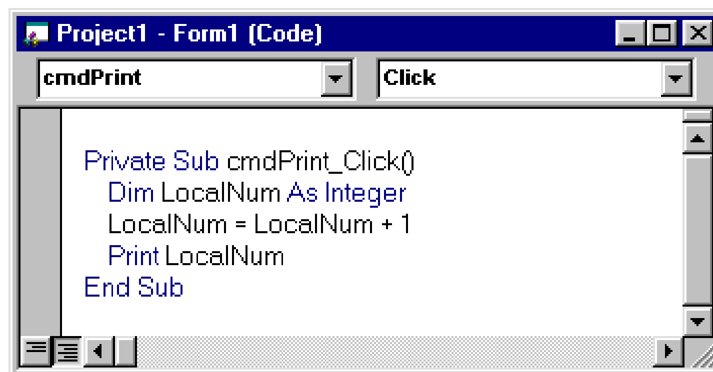
```
Static VisaCard As Long
```

לפנינו דוגמה אשר תמחיש את ההבדל בין משתנה מקומי לבין משתנה סטטי:

בתרשים 5.23 מוצהר המשתנה LocalNum כמשתנה מקומי מסוג Integer, והוא מאותחל באופן אוטומטי באפס. בשורה השנייה מקודם ערך המשתנה LocalNum באחד, וערכו עומד עתה על 1. בשורה האחרונה של השיגרה ערכו של המשתנה מודפס על פני הטופס בעזרת הפקודה Print.

השיגרה המודגמת בתרשים 5.23 מגיבה לאירוע Click של הלחצן cmdPrint. במילים אחרות, בכל פעם שנלחץ על הלחצן Print (תרשים 5.24) נקרא לשיגרה אשר תבצע את שורות הקוד שבה. כאשר נלחץ בפעם הראשונה על הלחצן, ייווצר המשתנה

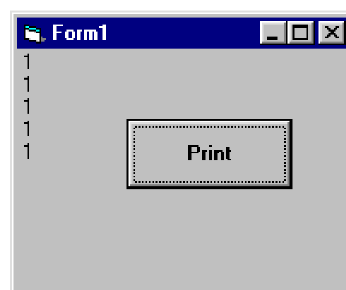
LocalNum ויוקצו לו מקום בזיכרון. משתנה זה יקבל אוטומטית את הערך אפס, ערכו יקודם ב- 1 ויוצג בטופס.



תרשים 5.23

אך מה יקרה אם נלחץ שוב על הלחצן Print? איזה ערך נקבל?

לכאורה, אנו אמורים לראות את הסיפרה 2 מוצגת בטופס, שהרי בקריאה הקודמת לשיגרה, כאשר לחצנו בפעם הראשונה על הלחצן קידמנו את ערך המשתנה ל-1. אולם בתרשים 5.24, המדגים חמש לחיצות על הלחצן Print, נראה שהערך 1 מוצג בכל פעם מחדש.

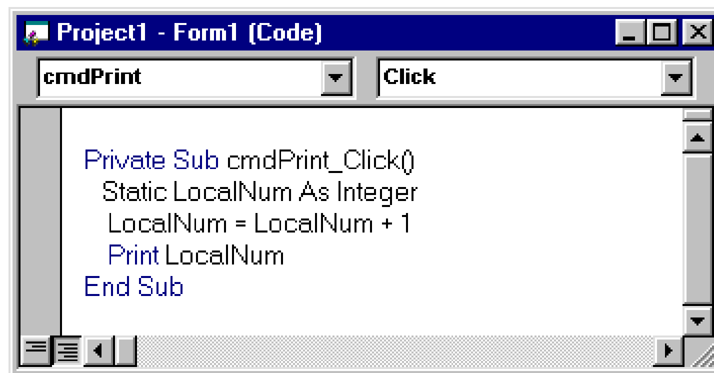


תרשים 5.24

הסיבה לכך היא שהמשתנה LocalNum הוא מקומי וככזה, בכל פעם שהשיגרה cmdPrint_Click() מסתיימת, מסתיים גם אורך החיים שלו, הוא משחרר את הזיכרון והערך שהכיל אינו נשמר. בלחיצה נוספת על הלחצן נגרום להיווצרות משתנה **חדש**, שיוקצו לו שוב מקום כלשהו בזיכרון. על משתנה חדש זה ששמו LocalNum יתבצעו שוב אותן פעולות: איפוס אוטומטי וקידום ערכו באחד. כך נבין מדוע בכל לחיצה יוצג שוב ושוב הערך 1 על גבי הטופס.

עתה נבצע את אותו תרגיל, אך הפעם נשתמש במשתנה סטטי.

בתרשים 5.25 מודגמת אותה שיגרה. הפעם מוצהר המשתנה LocalNum כמשתנה סטטי (Static). בלחיצה הראשונה על הלחצן Print, מאותחל המשתנה LocalNum באופן אוטומטי וערכו הוא אפס. ערכו מקודם ב- 1 ומוצג בטופס.

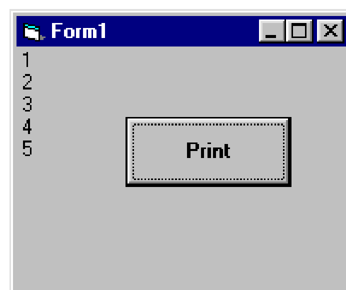


תרשים 5.25

אך מה קורה למשתנה כאשר מסתיימת השיגרה cmdPrint_Click()?

בסיום השיגרה המשתנה **אינו** משחרר את הזיכרון שהוקצה לו (למשתנה סטטי יש אורך חיים כאורך חיי המודול) ולכן הוא שומר על ערכו בתוך המודול. בלחיצה הבאה על הלחצן נקרא לשיגרה, והפעם יש למשתנה ערך (1) שהוא נושא עימו מהביצוע הקודם של השיגרה. לכן, כאשר נקדם את ערכו של המשתנה LocalNum ב-1, ערכו החדש יהיה עכשיו 2 וערך זה יוצג בטופס.

תרשים 5.26 מדגים חמש לחיצות על הלחצן Print כאשר המשתנה LocalNum סטטי.



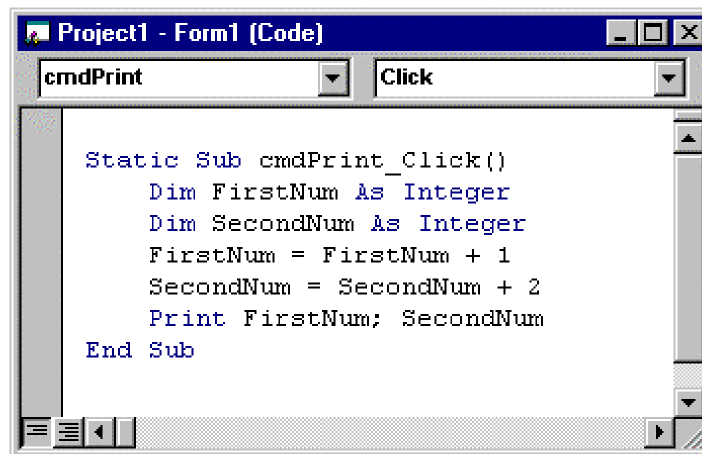
תרשים 5.26

פונקציה סטטית

יש אפשרות להצהיר על כל המשתנים שבתוך הפונקציה או בשיגרה כמשתנים סטטיים. לצורך זה נצהיר שהפונקציה או השיגרה הן מסוג "סטטי", ועל ידי זה כל המשתנים שבתוכן יוגדרו כסטטיים. השימוש במילת המפתח **Static** דרוש פעם אחת

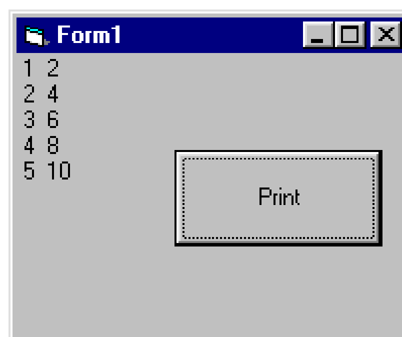
בלבד, לפני שם הפונקציה או לפני שם השיגרה, ואין צורך להצהיר על כל משתנה בנפרד שהוא סטטי.

אופן ההצהרה על כל המשתנים בפונקציה או בשיגרה כסטטיים נעשה על ידי הצבת מילת המפתח Static לפני שם הפונקציה או השיגרה (תרשים 5.27).



תרשים 5.27

בתרשים 5.27 מוצהרים המשתנים FirstNum ו- SecondNum כמשתנים רגילים, אך ההצהרה על השיגרה cmdPrint_Click() כסטטית הופכת את כל המשתנים בתוכה לסטטיים. ואכן בתרשים 5.28 נראה כי המשתנים FirstNum ו- SecondNum מתנהגים כמשתנים סטטיים (ולא לוקאליים). בכל לחיצה מתקדם ערך המשתנים FirstNum ו- SecondNum באחד ושניים בהתאמה.



תרשים 5.28

קבועים

משתנה קבוע (Constant variable) הוא מקום בזיכרון המאחסן ערך כלשהו. בניגוד למשתנה, לא ניתן לשנות את ערך המשתנה הזה, אשר קבוע לכל אורך התוכנית. לפיכך, לא תיתכן ביצוע פעולת השמה למשתנה קבוע, כי פעולה זו מחליפה ערך ישן בחדש, והדבר אינו אפשרי בקבוע.

לשימוש בקבועים במהלך התוכנית יתרונות רבים, וביניהם: קלות הקריאה של שורות הקוד ויכולת שינוי מהיר בשורות הקוד. בכתיבת הקוד משתמשים פעמים רבות בערכים מספריים, במחרוזות וכו', המייצגים ערכים משמעותיים שונים. לדוגמה, לציון הצבעה אפשר להשתמש בערך 1 כמייצג את הערך "בעד", 2 - מייצג "נגד" ו-0 - "נמנע". במקרה כזה, בכל מקום בו נרצה להשתמש בערכים אלה בשורות הקוד, קריא וברור יותר יהיה אם נשתמש בקבועים המייצגים אותם. לדוגמה:

```
Const Favor = 1
Const Against = 2
Const Abstain = 0
```

בנוסף, יכולת השינוי בקוד מהירה וקלה. נניח שאנו מפתחים תוכנית לניהול חשבונות ועלינו לחשב את המע"מ, שערכו הנוכחי 17%. מה יקרה אם יום אחד יוחלף ערכו של המע"מ ויעמוד על 18%? מכיון שהשינוי עלול להשפיע על שורות קוד רבות "המסתתרות" בין שורות קוד אחרות! ודאי היה יותר נוח לו היינו מגדירים את הקבוע הבא:

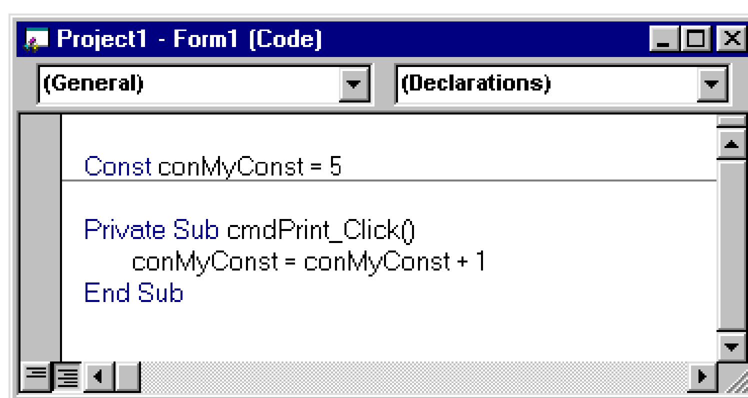
```
Const VAT = 17
```

ובכל מקום בו היה נדרש לחשב מע"מ, היינו משתמשים בקבוע **VAT**. ברגע שערך המע"מ יוחלף ל- 18%, עלינו לשנות שורת קוד אחת **בלבד**, שורת ההצהרה.

אופן הגדרת קבוע נעשה בעזרת מילת המפתח **Const**. לדוגמה:

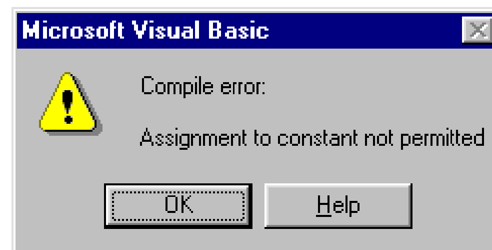
```
Const conMyConst = 5
```

בתרשים 5.29 מוגדר conMyConst כקבוע המכיל את הערך 5.



תרשים 5.29

בתוך השיגרה ()cmdPrint_Click נעשה ניסיון לקדם את ערכו של הקבוע באחד. פעולה זו אינה חוקית, כי היא מנסה לשנות ערך של משתנה קבוע. ואמנם, לחיצה על הלחצן Print הפעילה את האירוע ()cmdPrint_Click וגרמה להודעת שגיאת הידור כמודגם בתרשים 5.30.



תרשים 5.30

ערך הקבוע לא חייב להיות מספר; ניתן להגדיר קבוע מסוג מחרוזת, תאריך וכו'. לדוגמה:

```
Const conCountry = "Israel"
Const conBirthday = #21/11/70#
```

הקבוע יכול לאחסן ביטוי, גם אם הוא כולל קבוע אחר. לדוגמה:

```
Const conSum = intPrice * intQuantity
Const conSecond = conFirst + 2
```

בדומה למשתנה ברמת המודול, ניתן להגדיר קבוע כ- Public (במודול בלבד) ואז טווח ההכרה (ואורך החיים) שלו הוא לכל אורך חיי המודול (והתוכנית). לדוגמה:

```
Public Const conSum = intPrice * intQuantity
```

באותה מידה, הגדרת קבוע בתוך שיגרה או בתוך פונקציה גורמת לו להתנהג כמשתנה מקומי.

תרגול

1. הצהר על משתנה מסוג String וקלוט לתוכו שם של משתמש.
2. הצהר על משתנה גלובלי, אתחל אותו, והצג אותו מטופס אחר.

6: פקודות השמה

תפקיד המשתנים בתוכנית לאחסן ולשמור ערכים מסוגים שונים, בהם ערכים מספריים, מחרוזות, תאריכים ועוד, הכל בהתאם לסוג המשתנה. המשתנה שומר על ערכו לכל משך החיים שלו ואינו משנה אותו, אלא אם אנו משנים אותו במכוון על ידי פקודת השמה. תפקיד פקודת ההשמה "לשים" ערך במשתנה, ובמילים אחרות, להחליף ערך שמאוחסן בו בערך אחר רצוי, מעין "החלפה". **פקודת ההשמה** (Assignment instruction) נראית באופן כללי כך:

```
Variable = Value
```

לדוגמה:

```
Num = 3
Name = "Yoav"
```

קריאת פקודת השמה היא **מימין לשמאל**, ופירוש הדבר שהערך 3 יאוחסן במשתנה Num. כאן למשל,

```
Price = Num
```

הערך המאוחסן במשתנה Num יחליף את הערך הקיים במשתנה Price, ולא להיפך. בסיום פקודת השמה זו, Num ישמור על ערכו הנוכחי, ואילו ערכו הישן של Price יוחלף בערך שנמצא כעת במשתנה Num.

ניתן להשתמש בביטויים בפקודת השמה. הנה דוגמה:

```
ClientPrice = Price * (100 + Tax) / 100
```

גם במקרה זה, קודם מחושב **כל הביטוי** באגף ימין של פקודת ההשמה, ורק לאחר מכן מושמת **התוצאה** לתוך המשתנה ClientPrice.

אתחול משתנים

כל עוד לא מתבצעת פקודת השמה, לא משתנה ערכו הנוכחי של המשתנה שמשמאל לסימן השוויון. אך מהו ערך זה? מתי קיבל המשתנה את ערכו בפעם הראשונה?

השמת ערך במשתנה בפעם הראשונה נקראת "**אתחול**" (Initialization). מתכנת מקצועי לעולם לא ישתמש במשתנה מבלי לאתחל אותו תחילה. פקודת ההשמה Text1.Text=Num יכולה לגרום לנזק רב, אם אין יודעים מראש מה ערכו של Num. משתנה זה מכיל נתון בלתי ידוע, ובשפת העם - "זבל". במילים אחרות, אסור לנו להסתמך מראש על ערכו של המשתנה, אלא אם אנחנו יודעים בוודאות מהו. יש לאתחל את המשתנה לפני הפעם הראשונה שמשתמשים בו. הערך שבו נאתחל את המשתנה, משתנה לפי העניין, ובהנחה שהמשתנה מתפקד כמונה או כסוכֵם, נאתחל אותו בערך 0.

אתחול המשתנה נעשה בפקודת השמה ולפיכך, אתחול משתנה בערך 0 יתבצע כך:

```
Counter = 0
```

בוויז׳ואל בייסיק המשתנים מאותחלים באופן אוטומטי: משתנה מספרי מקבל ערך התחלתי אפס, משתנה מסוג Variant מאותחל בערך Empty ומשתנה מחרוזת מאותחל במחרוזת ריקה ("", zero-length string). עם זאת, מומלץ להקנות לעצמנו הרגל טוב, ולאתחל את המשתנים החשובים לנו לפני השימוש בהם.

בפרק הקודם למדנו את מילת המפתח **Const** המגדירה קבוע. אם נשים לב, גם בהגדרת קבוע אנו משתמשים בפעולת אתחול. משתנה קבוע הוא משתנה אשר מאותחל **פעם אחת** בלבד, מכיון שפקודת אתחול נוספת (שהיא פקודת השמה לכל דבר) אינה חוקית לגביו וגם אסורה.

ביטויים

ביטוי חשבוני (Arithmetic Expression) מורכב מאופרנד אחד או אופרנדים אחדים. כאשר הביטוי מורכב מאופרנדים אחדים, הם מופרדים על ידי אופרטורים חשבוניים.

- **אופרנד** הוא תא בזיכרון, קבוע או משתנה.
- **האופרטורים החשבוניים** הבסיסים הם: חיבור (+), חיסור (-), כפל (*) וחילוק (/). אופרטורים אלה הם אופרטורים חשבוניים לכל דבר, וסדר הקדימויות המתמטי, בו כפל וחילוק קודמים לחיבור וחיסור, נשמר גם כאן. ניתן, כמובן, להשתמש בסוגריים כדי להתערב ולשנות את סדר הקדימות של הפעולות.

דוגמאות לביטויים חשבוניים:

1. 6*5
2. 17-9
3. 144/12
4. 42+58
5. 9*(6+3)

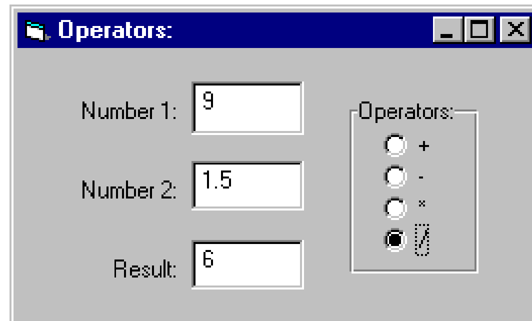
ביטויים חשבוניים מהווים גם כחלק מפקודות השמה, ולדוגמה:

1. Num = 9*3
2. Cost = Quantity*Price
3. Num1 = 100/(Num1+5)

את השימוש באופרטורים נתרגל בעזרת התוכנית הבאה:

1. צור את הטופס כדוגמת תרשים 6.1.

2. בנה את קבוצת הפקדים מסוג OptionButton כמערך פקדים.



תרשים 6.1

3. קבע את המאפיינים על פי טבלה 6.1.

טבלה 6.1

פקד	מאפיין	ערך
Label1	Caption	Number 1:
Label2	Caption	Number 2:
Label3	Caption	Result
Text1	Name Text TabIndex	txtNum1 (Empty) 0
Text2	Name Text TabIndex	txtNum2 (Empty) 1
Text2	Name Text	txtResult (Empty)
OptionButton	Name Caption	optOperator / ,* ,- ,+
Frame1	Caption	Operators:

4. כתוב את שורות הקוד האלו עבור האירוע Click של הפקד optOperator :

```
Private Sub optOperator_Click(Index As Integer)
   Dim num1 as single, num2 As Single
   num1 = Val(txtNum1.Text)
   num2 = Val(txtNum2.Text)
   Select Case Index
    Case 0      'Operator +
       txtResult.Text = num1 + num2
    Case 1      'Operator -
       txtResult.Text = num1 - num2
    Case 2      'Operator *
       txtResult.Text = num1 * num2
    Case 3      'Operator /
       txtResult.Text = num1 / num2
   End Select
End Sub
```

בחרנו באירוע Click דווקא, כי מטרת התוכנית להציג תוצאה רצויה בבחירת אחת מהאפשרויות, על ידי לחיצה על אחד הפקדים optOperator מסוג OptionButton. נשים לב לעובדה, כי השיגרה מקבלת את הפרמטר Index. הסיבה לכך היא שקבוצת הפקדים optOperator היא מערך.

בתוך השיגרה אנו מצהירים על שני המשתנים Num1 ו-Num2, אשר מקבלים את הערכים שהקשנו לתוך תיבות הטקסט txtNum1 ו-txtNum2 בהתאמה. מכיון שהקשנו את המספרים לתוך תיבת טקסט, עלינו להמיר אותם לערכים מספריים. הפונקציה אשר יודעת להמיר טקסט לערך מספרי, היא Val(). ואכן, את הערכים הכתובים בתוך תיבות הטקסט - המיוצגים על ידי המאפיין Text של הפקדים - המרנו לערך מספרי בעזרת פונקציה זו. התוצאה אוחסנה בעזרת פקודת השמה, בתוך המשתנים Num1 ו-Num2 בהתאמה.

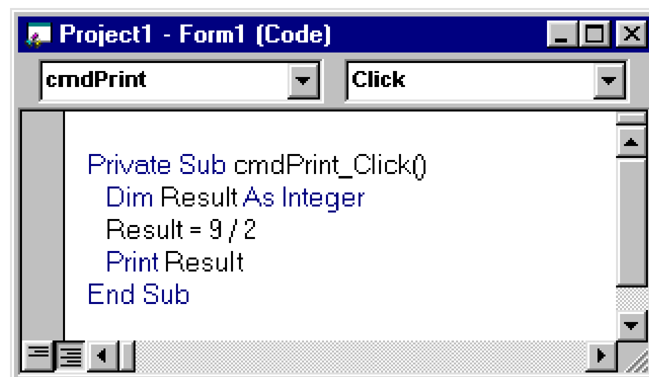
עתה עלינו לבדוק באיזו אפשרות בחרנו, ובאיזה אופרטור השתמשנו לצורך התרגיל. באמצעות מבנה ההחלטה **Select Case** (אשר כבר ראינו אותו בדוגמה קודמת ונדון בו שוב בפרק 8), אנו מבררים את ערכו של Index, ובמילים אחרות - מזהים את האפשרות שנבחרה. על פי אפשרות זו מופעל האופרטור על שני המשתנים Num1 ו-Num2. את התוצאה אנו מאחסנים בעזרת פקודת השמה, כערך של המאפיין Text של תיבת הטקסט txtResult, אשר גם מציג אותה בתיבה זו.

5. הרץ את התוכנית (F5). הקלד שני מספרים בשתי תיבות הטקסט ובחר באופרטור הרצוי. את התוצאה תוכל לראות בתיבת הטקסט התחתונה - Result.

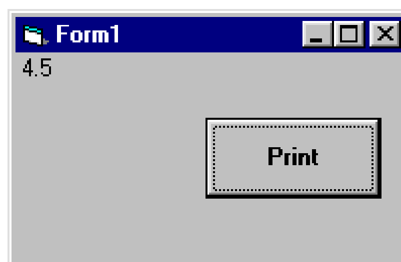
אופרטורים נוספים

מלבד ארבעת האופרטורים החשבוניים הבסיסיים שהכרנו עתה, ישנם עוד שלושה אופרטורים המאפשרים פעולות מתמטיות מגוונות.

- האופרטור " \ " - מבצע חילוק שלם. אם נחלק את המספר 9 ב-2 בעזרת אופרטור החילוק הרגיל " / " (תרשים 6.2), נקבל את התוצאה 4.5 (תרשים 6.3).



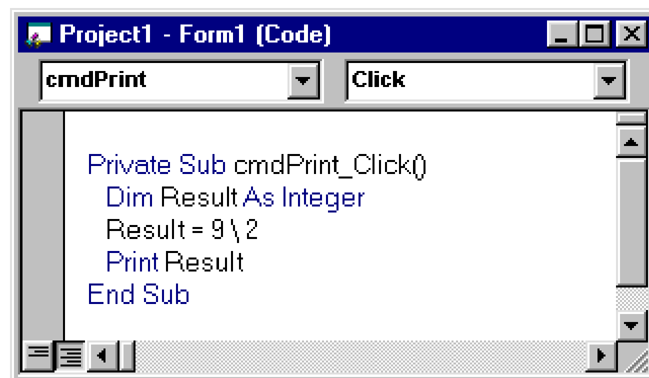
תרשים 6.2



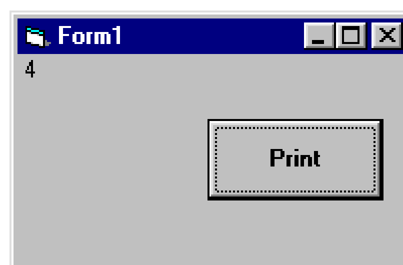
תרשים 6.3

אם החלק שאחרי הנקודה העשרונית בתוצאה אינו משמעותי עבורנו ונרצה לדעת רק את הערך השלם, נשתמש באופרטור חילוק " \ " (תרשים 6.4).

על כן נקבל הפעם: 9 \ 2 = 4 (תרשים 6.5).

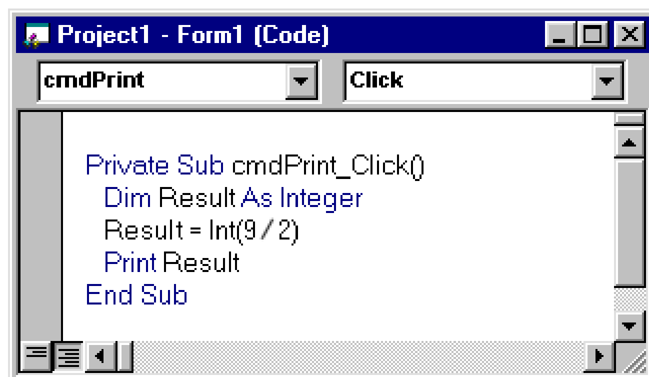


תרשים 6.4



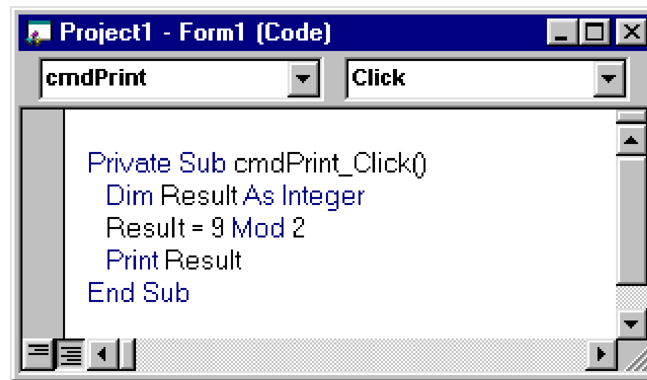
תרשים 6.5

שים לב שהפונקציה ()Int מבצעת את אותה פעולה (תרשים 6.6) כאשר האופרטור הוא אופרטור חילוק **רגיל,** הביטוי (9 / 2)Int יתן לנו את התוצאה 4.

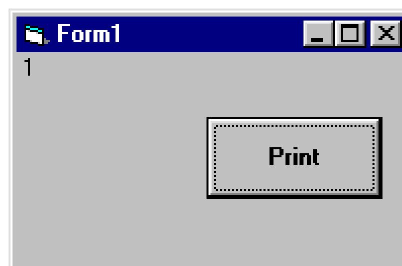


תרשים 6.6

• האופרטור **Mod** מציג את שארית החילוק (מודולו). תוצאת הביטוי "9 Mod 2" (תרשים 6.7) תהיה 1 (תרשים 6.8), מכיון שהמספר 4 נכנס פעמיים בצורה שלמה בתוך המספר 9, והשארית היא **1**.



תרשים 6.7

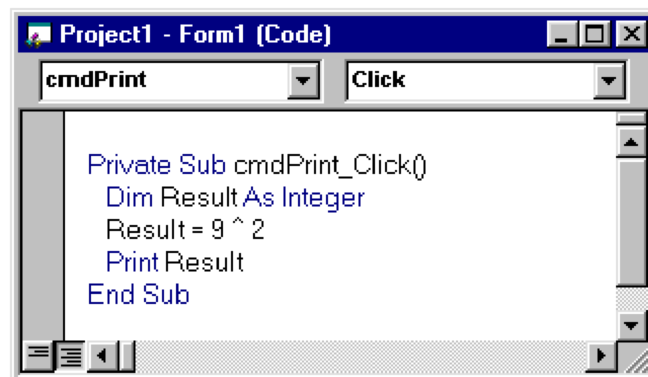


תרשים 6.8

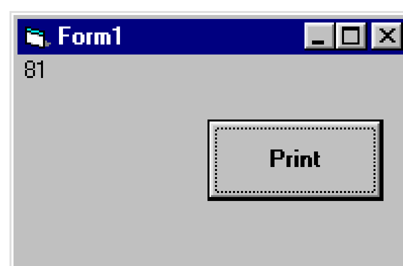
דוגמה נוספת: 20 Mod 6 = 2, שבה יש 3 כפולות שלמות של 6 והשארית היא 2.

• האופרטור " ^ " הוא אופרטור החזקה (תו זה מופיע בלוח המקשים במקש של הספרה 6). אופרטור החזקה מבצע את פעולת ההעלאה בחזקה המוכרת לנו. דוגמאות לשימוש באופרטור:

1. 9 ^ 2 = 81 (תרשימים 6.9 ו-6.10)
2. 3 ^ 3 = 27



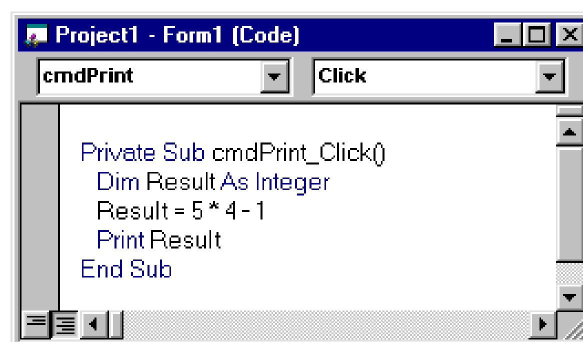
תרשים 6.9



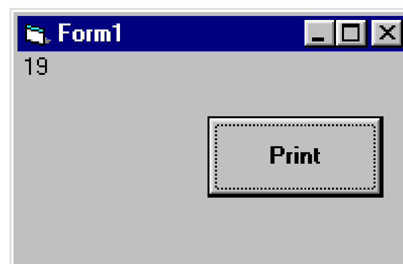
תרשים 6.10

סדר קדימויות

בביטויים חשבוניים רגילים ויזואל בייסיק מחשבת את תוצאת הביטוי בהתאם ל**סדר הקדימויות** (Precedence rules). לדוגמה, אופרטור הכפל קודם לאופרטור החיסור ולכן, בתרגיל המודגם בתרשים 6.11, תחילה מחושב הביטוי 4*5 ורק אחר כך מחסרים 1 מהתוצאה (תרשים 6.12).

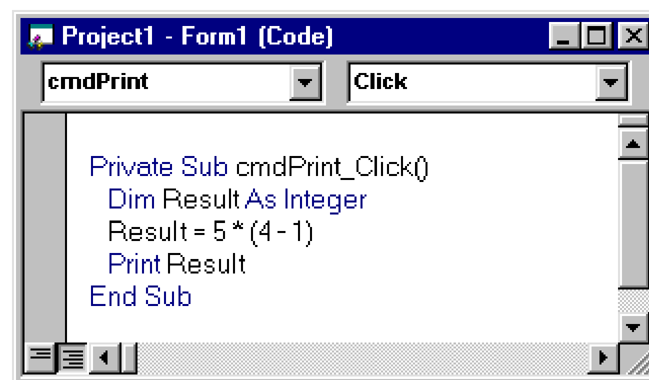


תרשים 6.11



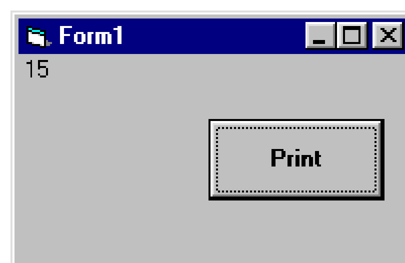
תרשים 6.12

שים לב, שלמרות סדר הקדימויות הסוגריים **קודמים** לכל. לכן, אם ניקח את הביטוי מתרשים 6.11 ונוסיף בו סוגריים (תרשים 6.13), יחושב תחילה הביטוי שבסוגריים ורק אחר כך יופעל אופרטור הכפל על התוצאה של הביטוי שחושב בסוגריים.



תרשים 6.13

במקרה זה התוצאה תהיה 15 ולא 19 (תרשים 6.14).



תרשים 6.14

להלן סדר הקדימויות:

1. סוגריים.
2. אופרטור החזקה (^).
3. אופרטור השלילה (-) .
4. אופרטור הכפל (*), אופרטור החילוק (/).
5. אופרטור החילוק השלם (\ ).
6. אופרטור השארית (מודולו) Mod.
7. אופרטור החיבור (+), אופרטור החיסור (-).

אופרטורי השוואה

הביטוי Num = 20 הוא למעשה פקודת השמה סטנדרטית. אופרטור השוויון (=) מייצג כאן פקודת השמה. במקרה זה, הערך 20 "מושם", או מאוחסן, בתוך המשתנה Num. אופרטור השוויון (=) מתפקד גם כאופרטור **השוואה** (Comparison). משמעות הביטוי Num = 20 במקרה זה: האם Num שווה ל-20. במילים אחרות, האם תוכנו, או ערכו של המשתנה Num הוא 20.

ההבדל בין השניים הוא, שבאופרטור השמה אנו מבצעים **פעולה** בעזרת אופרטור השוויון, ו-Num מקבל ערך חדש. לעומת זאת, כשאופרטור השוויון מתפקד כאופרטור **השוואה**, אנו שואלים **שאלה**. המטרה היא לקבל מידע בלבד, ולא מתבצעת כל פעולה על שני האופרנדים המרכיבים את הביטוי וערכם **לא** משתנה. לאחר מכן, אנו בוחנים את המידע אודות ההשוואה ופועלים לפיו. נראה זאת במשפט IF, למשל.

מלבד אופרטור השוויון (=) קיימים אופרטורי השוואה נוספים. המידע המתקבל מהביטוי המורכב בעזרת אופרטורי השוואה הוא **כן** או **לא, בלבד!** לחילופין, השאלה היא "האם הביטוי נכון (מתקיים), או שאינו נכון". לכן, הביטויים המורכבים מאופרטורי השוואה נקראים **ביטויי התניה**, או **ביטויים מותנים**. לדוגמה, האופרטור גדול (>) הוא אחד מאופרטורי ההשוואה, ובאמצעותו נדע אם אופרנד אחד בביטוי גדול מחברו, או לא. בהנחה שהביטוי Num>15 הוא אמת ("נכון"), נסיק שערכו של המשתנה Num גדול מ- 15 (את ערכו המדויק לא נוכל לדעת בדרך זו).

באופן דומה מתפקדים שאר אופרטורי ההשוואה המפורטים בטבלה 6.2.

טבלה 6.2

אופרטור	משמעות	דוגמה
=	שווה	Num = 20
<>	שונה	Price <> Num
>	גדול	Price > 35
<	קטן	5 < Val(txtNum.Text)
>=	גדול או שווה	500 >= Cost
<=	קטן או שווה	Grade <= 100

יש לשים לב לשתי נקודות חשובות בטבלה. בניגוד לפקודת השמה הנקראת מימין לשמאל, ביטוי מותנה נקרא **משני** הכיוונים כאחד. שני הביטויים המותנים Price > 50 ו- Sum = 17 תקפים. לעומת זאת, בפקודת השמה הביטוי השני Sum = 17 שגוי. עם זאת, חשוב לשים לב לאופרטור שמשתמשים בו, הביטוי A > B, **אינו** זהה לביטוי B < A כי בכל אחד מהם אנו בודקים יחס **שונה**!, בניגוד למקרה בו נעשה שימוש באופרטור ההשוואה A = B ; B = A בו שני המקרים זהים.

באופרטורי השוואה ובביטויים מותנים, נשתמש כחלק ממבני החלטה עליהם נלמד בפרק 8.

אופרטורים לוגיים

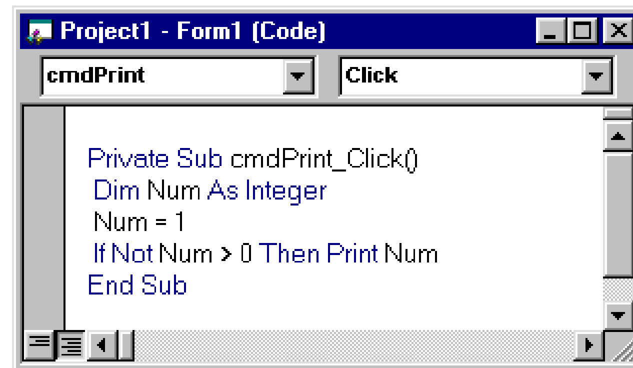
אופרטורים נוספים, השימושיים במבני החלטה, הם אופרטורים **לוגיים** (Logical operators). תפקידם להפוך ביטוי מותנה ל**ביטוי מותנה מורכב**. למעשה, ביטוי מורכב מכיל בתוכו מספר ביטויים מותנים המופרדים בעזרת אופרטורים לוגיים. ככלל, ביטוי מותנה מורכב מתקיים רק כשכל חלקי הביטוי מתקיימים. שלושת האופרטורים הלוגיים הם And, Or, Not.

- **האופרטור And** (**וגם**) קובע שרק כאשר **כל** חלקי הביטוי מתקיימים, הביטוי הוא "נכון" או "אמת". בביטוי Price > 80 And Quantity = 20, רק אם Price גדול מ- 80 **וגם** Quantity שווה ל- 20, יהיה הביטוי כולו אמת; אחרת הביטוי שקרי והוא אינו מתקיים.
- **האופרטור Or** (**או**) קובע שכאשר **לפחות אחד** מחלקי הביטוי "אמת" ("מתקיים") - **כל** הביטוי הוא אמת (למרות שחלקים אחרים אינם אמת).

בביטוי Price > 80 And Quantity = 20, אם Price גדול מ- 80 **או** Quantity שווה ל- 20, הביטוי כולו אמת. רק כאשר כל חלקי הביטוי אינם מתקיימים, הביטוי שקרי ואינו מתקיים.

• **האופרטור Not** (**לא**) קובע שהביטוי נכון רק כאשר **לא** מתקיים התנאי המרכיב אותו. בביטוי Not Quantity = 20, הביטוי יתקיים רק כאשר ערך המשתנה Quantity אינו 20. במילים אחרות, אם הביטוי המותנה הוא אמת, Not הופך אותו לשקרי; וכן להיפך, אם הביטוי המותנה הוא שקרי (לא מתקיים), Not הופך אותו לאמת.

בחן את שורות הקוד שבתרשים 6.15:



תרשים 6.15

לא למדנו עדיין להשתמש במבנה ההחלטה **IF**, אך נאמר שהוא בודק אם הביטוי המותנה מתקיים או שאינו מתקיים; ובמילים אחרות, אם הביטוי הוא אמת או שקר. אם התשובה חיובית (מתקיים, אמת), מתבצעת הפקודה שבאה לאחר המילה **Then**, ובמקרה שלנו - הפקודה Print. פקודה זו מדפיסה את הערך של Num. אם התשובה שלילית, התוכנית תמשיך לשורות הקוד הבאות אחרי משפט IF.

בשורות קוד אלו, אם הביטוי המותנה מתקיים, התוכנית מדפיסה בטופס את ערך המשתנה Num (שהוא 1); אם לא - התוכנית לא מבצעת דבר. השאלה היא, האם הביטוי מתקיים, ואז מילא מודפס המשתנה Num, או לא?

כדי לבחון את הביטוי Not Num > 0 נפרק את הביטוי לגורמים. הביטוי המותנה Num > 0 כשהוא לבד, הוא ביטוי אמיתי ונכון. ערכו של Num הוא 1 והוא אכן גדול מאפס. בשלב זה בא האופרטור Not והופך את תוצאת הביטוי. מכיון שהביטוי כשלעצמו אמת, צירוף האופרטור Not הופך אותו לביטוי שקרי, והתנאי אינו מתקיים. במקרה כזה, שהתנאי אינו מתקיים, הפקודה Print שלאחר המילה Then לא תתבצע, ולא יודפס דבר בטופס, והתוכנית תמשיך לשורת הקוד שאחרי המשפט IF.

טבלה 6.3 מסכמת את אפשרויות השימוש באופרטורים הלוגיים:

טבלה 6.3

אופרטור	**ביטוי מותנה א׳**	**ביטוי מותנה ב׳**	**ערך הביטוי המורכב כולו**
And	אמת	אמת	אמת
	אמת	שקר	שקר
	שקר	אמת	שקר
	שקר	שקר	שקר
Or	אמת	אמת	אמת
	אמת	שקר	אמת
	שקר	אמת	אמת
	שקר	שקר	שקר
אופרטור	**ביטוי מותנה**		**הביטוי המותנה כולו**
Not	אמת		שקר
	שקר		אמת

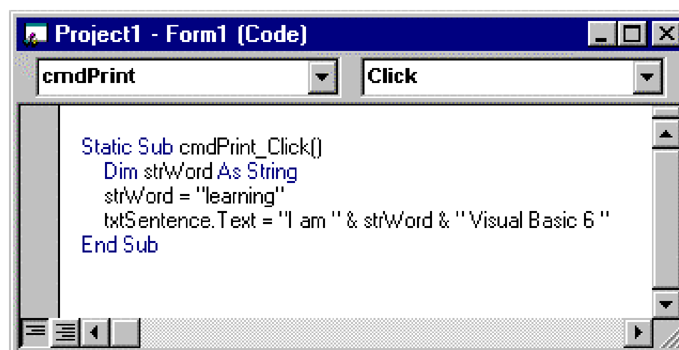
למתקדמים: ישנם עוד שלושה אופרטורים לוגיים Eqv ,Xor ו-Imp (המופעלים על ביטויים). ניתן לקרוא עליהם במערכת העזרה של ויז׳ואל בייסיק.

שרשור משתנים

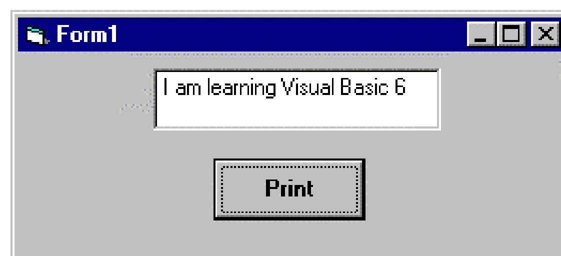
ראינו שניתן לבצע פעולת חיבור מתמטית פשוטה בעזרת האופרטור פלוס (+). אך מה קורה כאשר יש לנו שתי מחרוזות המיוצגות על ידי שני משתנים מסוג String, האם ניתן גם אותם לחבר? האם ניתן להרכיב מהמשתנה ”Fname=“David ומהמשתנה ”Lname = “Levi משתנה חדש אשר יכיל את שניהם, לדוגמה "Name = "David Levi?

התשובה היא **כן**. אופרטור **השרשור** & מבצע שרשור (concatenation) של מספר מחרוזות ויוצר מחרוזת אחת רציפה. את הרווח בין המחרוזות אנו משרשרים בעצמנו, כי האופרטור אינו יודע לעשות זאת.

תרשימים 6.16 ו- 6.17 מדגימים את השימוש באופרטור השרשור:



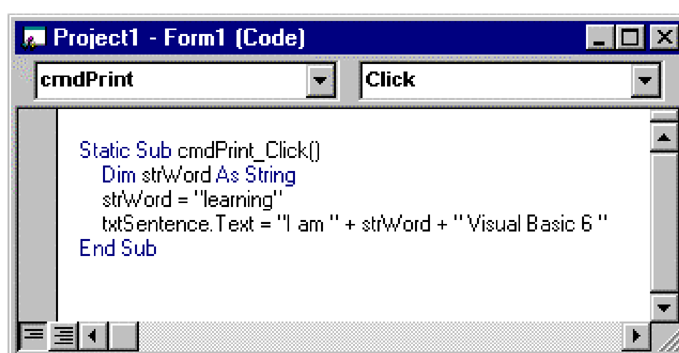
תרשים 6.16



תרשים 6.17

בדוגמה זו שרשרנו שלוש מחרוזות לתוך פקד אחד: "I am", "learning" (המיוצגת על ידי המשתנה strWord) ו-"Visual Basic." המשפט השלם אשר יופיע בפקד txtSentence הוא: "I am learning Visual Basic."

בוויז'ואל בייסיק ניתן להשתמש באופרטור החיבור החשבוני (+) גם לחיבור שתי מחרוזות. שורות הקוד המודגמות בתרשים 6.18 זהות לחלוטין לתרשים 6.16 ומפיקות את אותה התוצאה.



תרשים 6.18

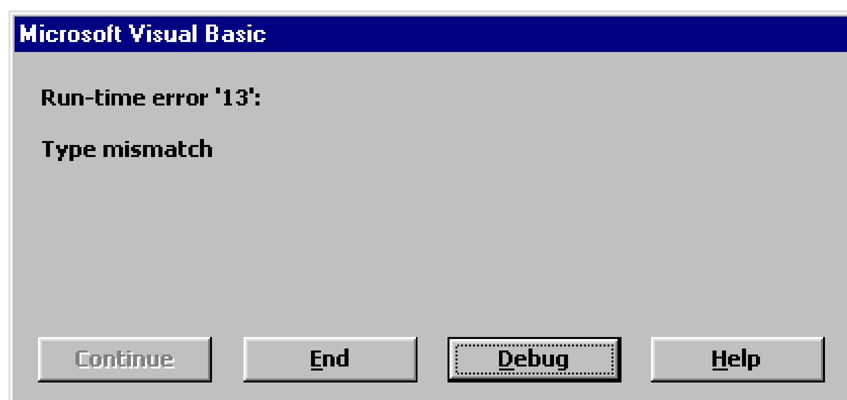
פונקציות לבדיקת משתנים

מטרת המשתנים בתוכנית לאחסן ולשמור ערכים מסוגים שונים. את הערכים מקבלים המשתנים באמצעות פקודות השמה, אך מיהו זה שקובע מה הערך שיושם במשתנה?

בדרך כלל משתנים מאותחלים בשורות הקוד על ידי המתכנתים. אולם פעמים רבות ערך המשתנה מתקבל כקלט מהמשתמש. לדוגמה, ערך המשתנה Age מכיל את גיל המשתמש, והוא התקבל מהמשתמש עצמו כאשר הקליד אותו לתיבת הטקסט txtAge. כדי שבמשתנה Age יתקבל ערך זה, עלינו לבצע את פקודת ההשמה:

```
Age = txtAge.Text
```

הכל טוב ויפה, אך כיצד נוכל לוודא שהמשתמש הכניס ערך מספרי המייצג את גילו, ולא הכניס בטעות את שמו? אם אכן כך יקרה, יהיה זה ה"באג" הראשון שלנו. התוכנית "תיכשל" ותודיע על שגיאה (תרשים 6.19) שמשמעותה - חוסר התאמה בין המשתנה (שהוא מסוג Integer) לבין הערך שברצוננו לאחסן בו (שהוא מחרוזת).



תרשים 6.19

נוכל להתחכם ולכתוב את פקודת ההשמה כך: Age = Val (txtAge.Text), ואז גם במקרה והמשתמש יקליד מחרוזת במקום מספר, התוכנית לא תיכשל. כמובן, שבכך לא נפתרה הטעות ואנו גם לא נדע עליה. הערך שיתקבל יהיה אפס (גיל לא הגיוני לכל הדעות) ואנו כמתכנתים, לא נהיה מודעים כלל לבעיה.

לאור האמור, נלמד להכיר ולהשתמש בפונקציות לבדיקת ערכי משתנים. בעזרתן ניתן לוודא שהערך אשר נאחסן במשתנה תואם את סוגו. נציג שלוש פונקציות: IsNumeric, IsDate ו- IsNull.

- הפונקציה **IsNumeric** מקבלת כפרמטר ביטוי או משתנה ומחזירה ערך אמת (True) כאשר הללו מוגדרים כמספר. אם הביטוי או המשתנה אינם מספר, הפונקציה תחזיר ערך שקרי (False).

בטופס המוצג בתרשים 6.20 מתבקש המשתמש לרשום את גילו ולאשר זאת בלחיצה על לחצן OK.

IsNumeric

Enter Your age:

OK

תרשים 6.20

כדי לוודא שהמשתמש הכניס ערך מספרי ולא את שמו, בטעות, נשתמש בפונקציה IsNumeric כמודגם בתרשים 6.21.

Project1 - Form1 (Code)

(General) (Declarations)

```
Private Sub cmdOK_Click()
  If Not IsNumeric(txtAge.Text) Then
    MsgBox "Enter numbers only!"
  End If
End Sub
```

תרשים 6.21

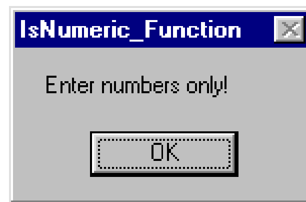
כאשר המשתמש יכניס בטעות את שמו (תרשים 6.22) ויאשר, תופיע תיבת הודעה המודיעה שעליו להקליד ערכים מספריים בלבד (תרשים 6.23).

IsNumeric

Enter Your age: Yossy

OK

תרשים 6.22



תרשים 6.23

- הפונקציה **IsDate** מקבלת כפרמטר ביטוי או משתנה ומחזירה ערך True כאשר הללו מוגדרים כתאריך. אם הביטוי או המשתנה אינם תאריך חוקי או לחילופין, לא קיים תאריך כזה (כדוגמת 31 בפברואר), הפונקציה תחזיר ערך False.

שורת הקוד

```
If Not IsDate (txtBirth.Text) then MsgBox "Enter appropriate date!"
```

תציג את הודעת השגיאה "Enter appropriate date!" כאשר התוכן שהמשתמש יקליד לתוך תיבת הטקסט txtBirth אינו תאריך חוקי, או תאריך שאינו קיים.

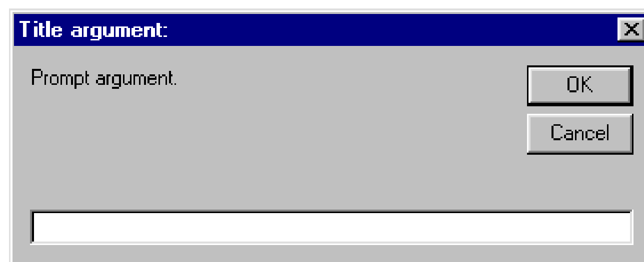
- הפונקציה **IsNull** מקבלת כפרמטר ביטוי או משתנה ומחזירה ערך בוליאני, המעיד אם הביטוי או המשתנה מכילים ערך Null או לא. הערך המוחזר True מצביע על כך, שהערך בביטוי הוא Null ; ערך מוחזר False מצביע ההיפך.

פקודות קלט ופלט

קלט ופלט בוויזואל בייסיק מתבצעים בדרך כלל באמצעות הפקדים Label ו-TextBox. בעזרת הפקד **Label** אנו מציגים למשתמש הודעות והוראות שונות, שאין באפשרותו לשנותן. בעזרת **TextBox** אנו קולטים נתונים מהמשתמש. פקדים אלה קבועים בטופס במהלך התוכנית ואינם משתנים. לפעמים נרצה להציג למשתמש הודעה זמנית כלשהי, כאשר הוא מקיש תאריך לא הגיוני, למשל. במקרה זה לא נציג את ההודעה בעזרת Label. כך גם ננהג בפעולת קלט זמנית שבה לא נשתמש בפקד TextBox.

תיבת קלט InputBox

תיבת הקלט InputBox (תרשים 6.24) מציגה חלון דו-שיח קטן למשתמש. בחלון זה יש תיבת טקסט, המאפשרת למשתמש להקליד ערך המוזן כקלט. תיבת הקלט מקבלת 7 פרמטרים, אך בהסבר זה נציג את הפרמטרים Prompt ו-Title, שהם החשובים והשימושיים יותר. למידע נוסף צריך להקליד InputBox בחלון Index שבמערכת העזרה.



תרשים 6.24

Prompt מייצג את ההוראה המוצגת בחלון הקלט, למשל: "הקלד נא את שמך:". מטרת ההוראה להודיע למשתמש מה אנו מצפים ממנו שיקליד (כקלט) לתיבת הטקסט שבחלון.

Title מייצג את כותרת חלון הקלט. מלבד ההוראה, ניתן לשכתב את כותרת החלון שבשורת הכותרת בחלק העליון של החלון.

תיבת הקלט מחזירה מחרוזת (String) המכילה את תוכן תיבת הטקסט שבחלון הקלט, לתוכה המשתמש הקליד את אשר רצה שנקלוט. מבנה פקודת הקלט הוא:

```
ReturnValue = InputBox(Prompt, Title)
```

לדוגמה,

```
Result = InputBox("Enter your age please:", "Age.")
```

כותרת החלון תהיה "Age" וההוראה למשתמש תהיה: "Enter your age please:". בהנחה שהמשתמש בגיל 25 ואת מספר זה הוא הקיש לתוך תיבת הקלט, המשתנה Result יקבל את הערך 25.

את הפרמטרים Prompt ו-Title יכולים להחליף משתנים מסוג String המאחסנים בתוכם את המחרוזות הרצויות.

תיבת פלט MsgBox

תיבת הפלט דרושה לצורך הודעה זמנית למשתמש. הודעות כדוגמת שגיאות (כתוצאה מהכנסת נתון שגוי על ידי המשתמש), אזהרות, הנחיות וכו', הן הודעות זמניות. במקרים אלה נשתמש בתיבת הפלט. אחרי קריאת ההודעה, המשתמש יכול לסגור את התיבה בלחיצה על הלחצן המתאים. דוגמה לתיבת פלט מוצגת בתרשים 6.25.



תרשים 6.25

התחביר של הפקודה להצגת תיבת פלט הוא:

```
MsgBox  Prompt, Buttons, Title
```

Prompt - מייצג את ההודעה אשר תוצג בחלון. ניתן להשתמש במשתנה מסוג String המכיל את המחרוזת הרצויה.

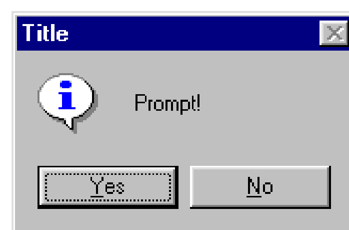
Buttons - מייצג את סוג תיבת ההודעה. יש סוגים שונים של תיבות: להודעה, לאזהרה, לשאלה, להצגת לחצני אישור וביטול, להצגת הודעה עם לחצן אישור בלבד, ועוד. סוג החלון נקבע על ידנו על פי תוכן ההודעה ועל פי אפשרויות התגובה שאנו מציעים למשתמש. סוגי התיבות הרבים מוצעים לפנינו באופן אוטומטי, ועלינו רק לבחור את סוג תיבת ההודעה הרצויה.

ניתן לבחור מספר סוגי לחצנים בתיבת פלט אחת, בין רשימת הלחצנים נשרשר את אופרטור החיבור "+". לדוגמה:

```
MsgBox ("Prompt!"), vbInformation + vbYesNo, "Title"
```

Title - מייצג את כותרת תיבת הפלט.

תרשים 6.26 מציג את תיבת הפלט, אשר שורת הקוד שלה הודגמה בשורת הקוד הקודמת:



תרשים 6.26

מלבד קבלת פרמטרים, תיבת ההודעה גם מחזירה ערך. ניקח לדוגמה את תיבת ההודעה המוצגת בתרשים 6.26. כיצד אנו, כמתכנתים, נדע אם המשתמש לחץ על Yes או על No. לצורך זה נשתמש בערך המוחזר של הפונקציה, אשר מציין את בחירת המשתמש, ואשר יכוון את פעולתנו. שורות הקוד הבאות נעזרות במבנה ההחלטה **If...Then**, במציאת הערך המייצג את בחירת המשתמש (שימו לב לסוגריים המוספים במקרה זה).

```
Private Sub Form_Click()
  Dim Result
  Result = MsgBox("Prompt!", vbInformation + vbYesNo, "Title")
  If Result = vbYes Then
     Print "Yes"
  Else
     Print "No"
  End If
End Sub
```

את מבנה המשפט If-Then-Else נלמד בפרק 8, אך כאן נסביר את הפעולה בקצרה: במשפט IF אנו בודקים אם תנאי מסוים תקף, או לא. כאשר התנאי תקף, והתשובה על הבדיקה היא "אמת", תתבצע הפעולה שכתובה אחרי המילה Then; אך אם התנאי אינו תקף והתוצאה היא "שקר", מתבצעת הפעולה שאחרי המילה Else. משפט If מורכב כזה יש לסיים במשפט End If.

הבה נחזור לדוגמה: במקום הקבוע vbYes, אשר מייצג לחיצה על הלחצן Yes, ניתן להשתמש בערך המספרי 6, המקביל לו. השורות הבאות מבצעות פעולה זהה לחלוטין לזו שמבצעות שורות הקוד הקודמות:

```
Private Sub Form_Click()
   Dim Result
   Result = MsgBox("Prompt!", vbInformation + vbYesNo, "Title")
   If Result = 6 Then
     Print "Yes"
   Else
     Print "No"
   End If
End Sub
```

גם לשאר הקבועים יש ערכים מספריים מקבילים:

Constant	Value	Description
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes
vbNo	7	No

תיבת הודעה מותאמת אישית

ראינו שוויז'ואל בייסיק מאפשרת תיבות הודעה מסוגים שונים. אך מה יקרה כשנרצה להציג למשתמש תיבה במבנה שונה, אשר לא מופיעה כחלק מההיצע של ויז'ואל בייסיק? במקרה כזה, נבנה תיבת הודעה מותאמת אישית.

בניית תיבת הודעה היא פעולה פשוטה מאוד, מכיון שניתן לראות אותה כטופס **רגיל** לכל דבר. השוני ביניהם הוא בשתי יכולות בלבד. אם נשווה בין השניים נמצא, כי בתיבת הודעה לא מופיעה תיבת הבקרה שנמצאת בצד ימין עליון של טופס רגיל. בנוסף, נמנעת מהמשתמש היכולת לשנות את גודלה, אשר נשאר קבוע.

מעיון ברשימת המאפיינים לטופס רגיל נמצא את שני המאפיינים הבאים: BorderStyle ו- ControlBox. במאפיין **BorderStyle** אנו קובעים את סוג המסגרת

של החלון. ערך ברירת המחדל של מאפיין זה הוא Sizable - 2, דהיינו, בעל יכולת שינוי גודל. קביעת הערך Fixed Single - 1 למאפיין זה תמנע מהמשתמש את היכולת לשנות את גודלו. חצי מהדרך עברנו בהפיכת טופס רגיל לתיבת הודעה.

המאפיין השני **ControlBox** קובע אם בשורת הכותרת של החלון תופיע תיבת הבקרה או לא. הערך True (ברירת המחדל) מציג את תיבת הבקרה והערך False מסתיר אותה. ניתן להסיר את לחצן "הגדל" שבתיבת הבקרה בלבד, על ידי קביעת ערך False למאפיין MaxButton. באותו אופן ניתן להסיר את לחצן "מזער" מתיבת הבקרה על ידי קביעת ערך False למאפיין MinButton.

הנה, גם על הבדל זה התגברנו ועתה לפנינו טופס "רגיל" (עם שינויים קטנים) שעבורנו הוא תיבת הודעה. כל שנותר לנו הוא להציב את הפקדים הרצויים על פני הטופס, את הלחצנים ואת שורות הקוד המתאימות.

תרשים 6.27 מדגים תיבת הודעה מותאמת אישית.



תרשים 6.27

השירות Show

תיבת הודעה היא טופס לכל דבר, וכדי להציגה נשתמש באחד משירותי הטופס, **Show**. שירות זה טוען את הטופס לזיכרון ומציג אותו. לדוגמה:

```
frmMain.Show
```

בדרך כלל, כאשר טופס מוצג בפני המשתמש, הוא יכול ל"התעלם" ממנו ולהפנות את המיקוד לטופס אחר המוצג במסך. אולם במקרים מסוימים לא נרצה שהמשתמש "יתעלם" מהטופס, ונמנע את העברת המיקוד אל טפסים אחרים שמוצגים, כל עוד המשתמש אינו סוגר את הטופס הממוקד. דוגמה נפוצה לכך היא תיבת הודעה המוצגת למשתמש לאזהרה, למשל. במקרה זה, כל עוד המשתמש אינו נותן אישור בלחיצה על **אישור** ואינו סוגר את התיבה, הוא לא יוכל להפנות מיקוד אל טופס אחר.

כאן נכנס לתמונה הפרמטר שמועבר על ידי התוכנית אל השירות Show. פרמטר זה יכול להיות אחד משני הערכים האלה:

1. **vbModal**, או הקבוע **1** החופף לו. במקרה זה הטופס אינו יכול לאבד מיקוד והמשתמש אינו יכול לפנות לטופס אחר באותו יישום, כל עוד הוא אינו סוגר את הטופס הממוקד. לדוגמה:

```
frmMain.Show 1
```

2. **ללא פרמטר** (ברירת המחדל), או הקבוע **0**. במקרה זה ניתן להעביר את המיקוד אל טופס אחר ביישום, אף בלא סגירת הטופס הממוקד.

שים לב, כי גם כאשר הפרמטר הוא vbModal, המונע מהמשתמש גישה לטופס אחר ביישום כל עוד הטופס הממוקד לא סגור, המשתמש יכול לגשת אל יישום **אחר** הפועל ברקע.

כאשר יש שורות קוד הבאות לאחר שורת הקוד המציגה את הטופס (על ידי Show), לדוגמה:

```
Sum Main()
   frmPassword.Show 1
   If Pass = True Then
     frmMain.Show
   Else
     End
   End If
End Sub
```

ביצוע שורות קוד אלו תלוי באופן פתיחת הטופס. כאשר הטופס נפתח במצב "מודאלי" (1), שורות הקוד הבאות אחריו לא תבצענה כל עוד הוא לא נסגר. אך כאשר הטופס נפתח במצב "לא מודאלי" (0), שורות הקוד הבאות אחרי משפט הפתיחה מתבצעות והתוכנית ממשיכה כרגיל, גם במקרה שהמשתמש אינו סוגר או משתמש בטופס הממוקד. בדוגמה שלנו, הטופס frmPassword נפתח במצב "מודאלי" (1) ולכן, כל עוד הטופס לא ייסגר, התוכנית לא תמשיך בפעולתה.

השירות Show מבצע למעשה שתי פעולות, טעינת הטופס לזיכרון והצגתו למשתמש. ניתן לטעון את הטופס ברקע בלי להציגו. פעולה זו מתבצעת בעזרת המשפט **Load**. לדוגמה:

```
Load frmMain
```

משפט זה מקבל כפרמטר את שם (המאפיין Name) של הטופס וטוען אותו לזיכרון ללא הצגתו למשתמש. היתרון בטעינת הטופס מראש לזיכרון בלבד הוא במהירות הופעתו כאשר המשתמש מבקש להציגו על ידי לחיצה על לחצן מתאים, או בבחירת אחת האפשרויות שבתפריט וכו'. במקום שטעינת הטופס לזיכרון תבוצע רק כאשר המשתמש יבקש זאת ויהיה עליו להמתין, הטופס כבר טעון מראש בזיכרון (כשהמשתמש לא מודע לכך) והצגתו תהיה מהירה.

שחרור טופס מהזיכרון (וממילא, סגירתו) מתבצע בעזרת המשפט **Unload** עם שם הטופס כפרמטר. אם הטופס המשוחרר מהזיכרון הוא הטופס הנוכחי, ניתן להשתמש במילה השמורה **Me** המייצגת את הטופס הנוכחי. לדוגמה:

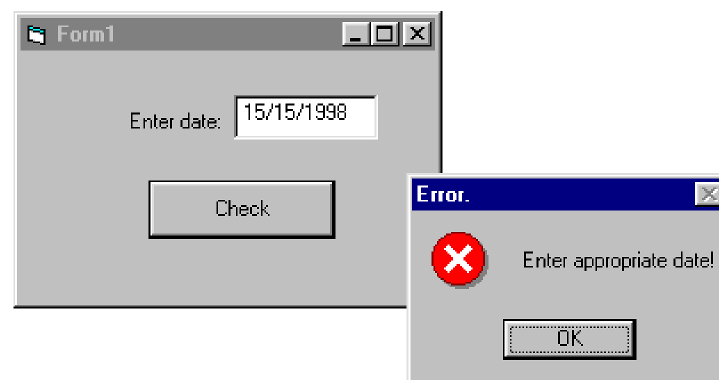
```
Unload Me
```

שים לב שניתן לסגור טופס ולהעלימו מעיני המשתמש, אך לא לשחררו מהזיכרון. פעולה זו מתבצעת בעזרת השירות **Hide**. לדוגמה:

```
Me.Hide
frmMain.Hide
```

תרגול

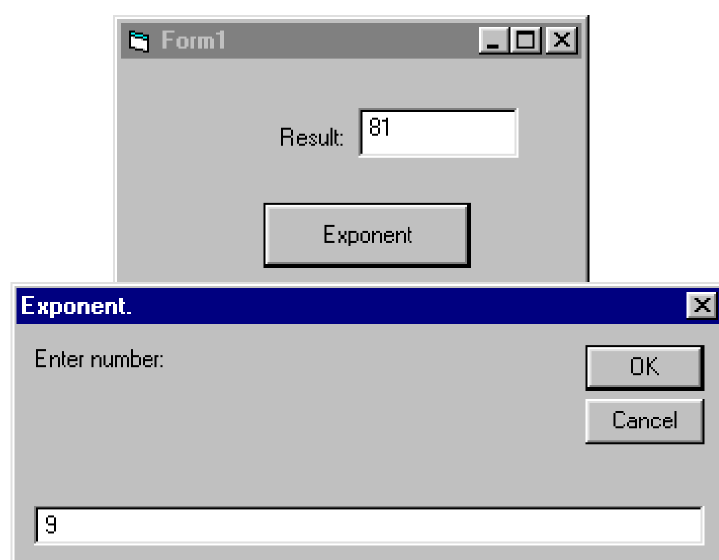
1. צור טופס המכיל שני פקדים, TextBox ו- CommandButton. הקש תאריך לתוך תיבת הטקסט ולחץ על הלחצן. אם התאריך שגוי (היעזר לצורך כך במבנה ההחלטה If...Then, ובפונקציה IsDate) הצג הודעה מתאימה (בעזרת MsgBox).



תרשים 6.28

2. צור טופס ועליו לחצן ותיבת קלט. בלחיצה על הלחצן תופיע תיבת קלט המבקשת מהמשתמש להכניס מספר שהמחשב יעלה בחזקת 2. כאשר המשתמש יאשר את אשר הקליד, תופיע התוצאה המבוקשת בתיבת הטקסט.

   שים לב שתיבת הקלט צריכה להחזיר ערך.



תרשים 6.29

7: מערכים

רבות משורות הקוד בתוכנית מיועדות לאחסון נתונים במשתנים, ועל כן גם נראה שמספר המשתנים בתוכנית הוא רב. כל משתנה מקבל שם **ייחודי** ומשמעותי לתפקידו בתוכנית. **המערך** (Array) לעומת זאת, מאפשר להתייחס למספר רב של משתנים **באותו** שם. המייחד את המשתנים במערך הוא **האינדקס** (Index), או במילים אחרות ציון המיקום היחסי של המשתנים במערך. משתנה במערך נקרא **איבר** (Element).

המערך מכיל איברים (הם המשתנים), שכל אחד מהם מחזיק ערך שלא בהכרח זהה לערך של איבר אחר, ולרוב הוא שונה. המשותף לכל איברי המערך הוא ש**כולם** מאותו סוג. במערך של משתנים מספריים, למשל, כל איברי המערך הם משתנים מספריים מאותו סוג (Integer, Long וכו'). ההבדל בין המשתנים הוא במיקומם במערך. לכל איבר במערך יש אינדקס המציין את מיקומו היחסי המדויק ביחס לראשית המערך, והפנייה למשתנה תהיה על ידי ציון אינדקס זה. אינדקס האיבר הראשון הוא אפס (0), אינדקס האיבר השני הוא 1, של השלישי - 2 וכן הלאה. לפיכך, את האיבר החמישי במערך Talmidim נציין כך: Talmidim(4).

אחד היתרונות שבשימוש במערך משתנים במקום מספר זהה של משתנים בודדים הוא בשם הזהה של כל המשתנים במערך. פנייה לשם אחד תוך שינוי מספר האינדקס שלו בלבד הרבה יותר קלה ומהירה. בעזרת לולאה ניתן לבצע פעולה אחת החוזרת על עצמה עם אותו שם משתנה ובשינוי האינדקס בלבד; ובמקרה שיש 100 משתנים נסתפק בשלוש שורות קוד במקום מאה! (נסה זאת לאחר שתלמד בסעיפים הבאים).

אופן ההצהרה

הצהרה על מערך דומה להצהרה על משתנה רגיל. בשני המקרים יש לציין את שם המשתנה וסוגו. אולם במערך, יש לציין גם את מספר איברי (משתני) המערך. לדוגמה:

```
Dim Name(10) As String
Dim Sum(13) As Long
```

בדוגמה זו הצהרנו על שני מערכים, האחד מסוג String והשני מסוג Long. המספר בסוגריים מציין את האינדקס המקסימלי במערך, החל מאינדקס 0 וכלה באינדקס המצוין בסוגריים. בדוגמה זו, במערך Name יש 11 איברים (משתנים מסוג String) שהראשון הוא בעל אינדקס 0 והאחרון בעל אינדקס 10. באותו אופן, למערך Sum יש 14 איברים.

המשפט Option Base

ברירת המחדל בוויז'ואל בייסיק היא אינדקס ראשון 0 במערך. לכן, כדי לבנות מערך בן 11 איברים (או משתנים) נרשום, לדוגמה, את ההצהרה הזו:

```
Dim Name(10) As String
```

בעזרת המשפט **Option Base** ניתן לשנות את ערך האינדקס המינימלי ל-1 (במקום 0, לפי ברירת המחדל) משפט זה מקבל פרמטר 0 או 1 המורה על ערך האינדקס המינימלי. כל עוד אין מציינים ערך מינימלי מיוחד, הערך שנקבע במשפט Option Base הוא אשר יחול במערך.

שורת הקוד הזו קובעת את ערך האינדקס התחתון ל- 1:

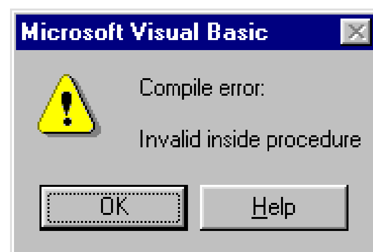
```
Option Base 1
```

ההצהרה הזו

```
Dim Name(10) As String
```

תגרום לבניית מערך בן 10 איברים בלבד, החל מאינדקס 1 (המייצג את האיבר הראשון) וכלה באינדקס 10 (לאיבר האחרון).

שים לב שעל המשפט Option Base ניתן להצהיר ברמת הטופס או המודול בלבד, בחלק **ההצהרות הכלליות** (General Declaration). שימוש במשפט Option Base בתוך שיגרה או פונקציה יגרום להודעת שגיאה (תרשים 7.1).



תרשים 7.1

קביעת גבולות למערך

כל עוד לא צוין אחרת, טווח ערכי האינדקסים נע מ- 0 (או 1 אם הוגדר כך במשפט Option Base, והדבר אפשרי) ועד לערך המצוין בסוגריים. עם זאת, ניתן לשנות ערכים אלה ולהגדיר במפורש את טווח ערכי האינדקסים. הגדרה זו נעשית בזמן ההצהרה על המערך.

לדוגמה:

```
Dim Num(1 To 20) As Integer
Dim Counter(30 To 50) As Long
```

בדוגמה זו הוגדרו הגבול העליון והתחתון של האינדקסים במערך על ידי המשתמש, אשר קבע שהמערך Num יכיל 15 איברים, כאשר הגבול התחתון שלו הוא 1 והעליון - 15 (גם כאשר לא השתמשנו במשפט Option Base). המערך Counter יכיל 20 משתנים עם טווח תחתון 30 וטווח עליון 50.

השמת ערכים

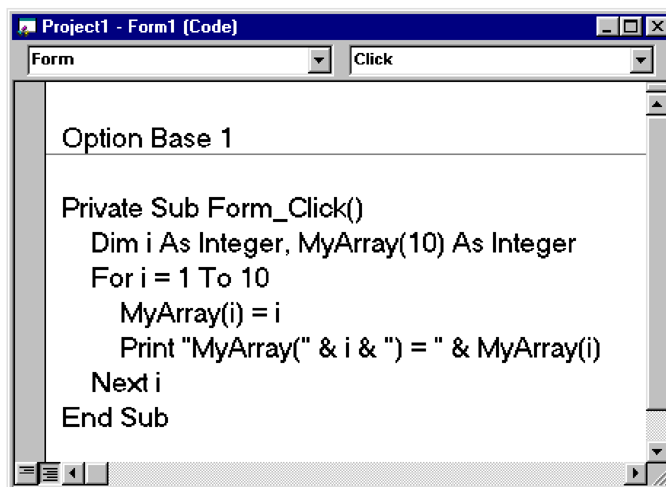
כאשר מצהירים על מערך בעזרת Dim, אנו יוצרים מערך ריק מערכים (מאותחל ל-0 במשתנים מספריים או ב-null למערך מסוג variant, או במחרוזת ריקה למערך מסוג string). במהלך התוכנית או בתחילתה, משמש המערך לאגירת ערכים. אופן השמת הערכים נעשה בפניה למערך בשמו, וציון האינדקס או האיבר במערך, שאליו אנו פונים. לכן, אם נרצה לשים את הערך “Dany” באיבר החמישי במערך Employees (ובהנחה שהגבול התחתון של המערך הוא 0 לפי ברירת המחדל) נבצע זאת כך:

```
Employee(4) = "Dany"
```

פנייה לאיבר הראשון תיראה כך:

```
Employee(0) = "Moshe"
```

השמת ערך לתוך איבר מערך היא פקודת השמה רגילה, וחלים עליה כל הכללים הרלוונטיים לפקודה מסוג זה. תרשימים 7.2 ו- 7.3 מציגים אתחול מערך והדפסתו, במבט קוד ובמבט ריצה בהתאמה.



תרשים 7.2



תרשים 7.3

מטריצות

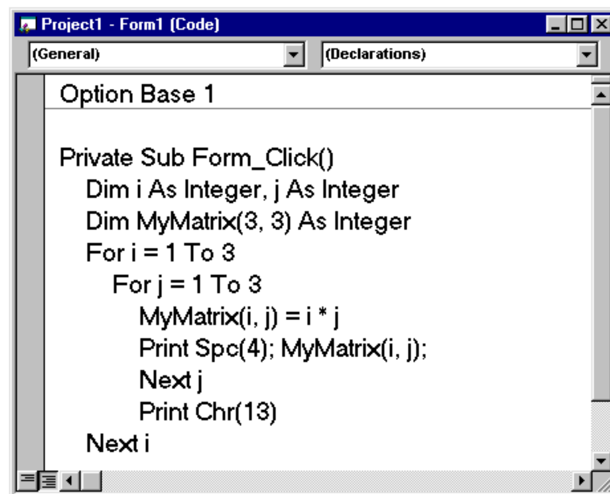
עד כה דנו במערכים חד-מימדיים. אולם ויז׳ואל בייסיק מאפשרת לנו להגדיר מטריצות של שני מימדים (מערכים דו-מימדיים) ויותר. הנה לדוגמה מערך דו-מימדי:

```
Dim Matrix (10, 10) As string
```

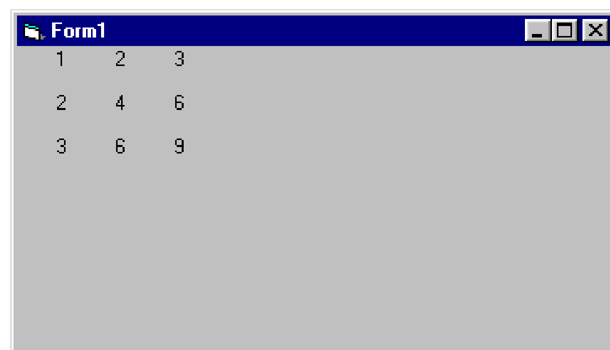
במקרה זה, כל מימד מכיל 11 איברים (0-10), ובכל המטריצה יהיו 121 איברים (11x11). אנו רואים שמערך דו-מימדי הוא למעשה טבלה, אשר מימד אחד שלו מייצג את השורות והמימד השני - את העמודות. בדומה למערך חד-מימדי, גם במערכים בעלי מספר מימדים ניתן לקבוע את הגבול העליון ואת הגבול התחתון. לדוגמה:

```
Dim Matrix(1 To 10, 1 To 10) As string
Dim Multi(4, 20 To 30, 2 To 7)
```

תרשימים 7.4 ו- 7.5 מציגים שימוש במערך דו-מימדי להדפסת לוח הכפל במטריצה 3X3.



תרשים 7.4



תרשים 7.5

הפונקציה Array()

בתרשים 7.4 הוצג אתחול מערך בעזרת לולאה. אפשרות נוספת ליצור מערך וגם לאתחל אותו בערכים רצויים היא על ידי הפונקציה **Array()**. פונקציה זו מקבלת כפרמטרים רשימה של ערכים, ובונה אותם כמערך (מסוג Variant). תחביר הפונקציה:

```
Array (arglist)
```

המילה arglist מייצגת את רשימת הארגומנטים הנשלחים לפונקציה.

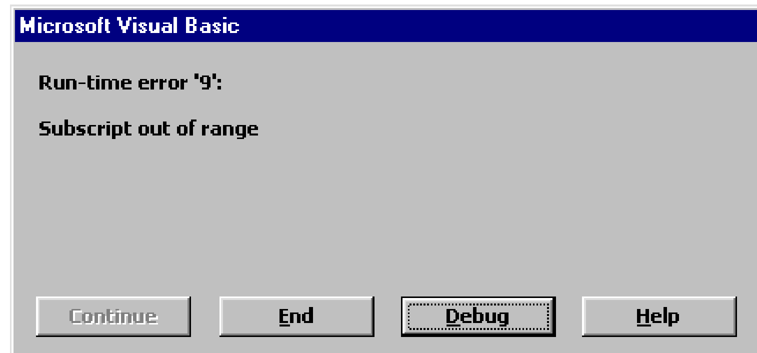
לדוגמה:

```
Dim MyArray
MyArray = Array(1, 2, 3, 4, 5)

Dim SecArray
SecArray = Array("Dan", "Ben", "Yossy", "David")
```

חריגה מתחום המערך

לכל מערך יש טווח ערכים המייצגים את מספר איברי המערך. טווח זה נע מהגבול התחתון, או במילים אחרות האינדקס ההתחלתי שלו, לבין הגבול העליון (האינדקס העליון). חריגה מטווח זה יוצרת מצב שגיאה, אשר מפסיק את פעולת התוכנית במתן הודעת שגיאה המוצגת בתרשים 7.6:



תרשים 7.6

הודעת השגיאה מציינת שהיתה חריגה מתחום המערך כתוצאה מפקודת השמה לאיבר במערך, אשר ערך האינדקס שלו חורג מהטווח המותר. חריגה זו יכולה לגרום להשמת ערכים בתאי זיכרון, שאינם חלק מהמערך או מיועדים לו, ולדריסת מידע באותם תאי זיכרון אשר נעשה בהם שימוש על ידי מערכת ההפעלה. כתוצאה מכך, יכול היישום ל"תקוע" את פעולת שאר היישומים הפועלים במקביל ליישום. לדוגמה:

```
Dim MyArray(10) As Integer
MyArray(11)=7
```

שגיאה זו קשה למעקב מכיון שהיא שגיאה לוגית. כאשר ויז'ואל בייסיק מהדרת את התוכנית, היא אינה יכולה להתריע על החריגה הזו, אולם בעת פעולת התוכנית, התוכנית מופסקת כאשר מתבצעת החריגה. בדוגמה זו נרשם MyArray(11) והחריגה ברורה, אך מה יהיה כשנכתוב MyArray(Serial), והמשתנה Serial, המחושב במהלך התוכנית, יקבל ערך גדול מ- 10?

כדי לזהות חריגה זו נשתמש בפונקציות UBound ו- LBound, אשר בעזרתן ניתן לבחון את גבולות המערך, ולמנוע חריגה מגבולותיו.

הפונקציה Ubound

הפונקציה Ubound מקבלת שני פרמטרים, את שם המערך ואת המימד הנבדק, ומחזירה ערך Long המציין את הגבול העליון של אותו מימד במערך.

תחביר הפונקציה הוא:

```
UBound(arrayname[, dimension])
```

לדוגמה:

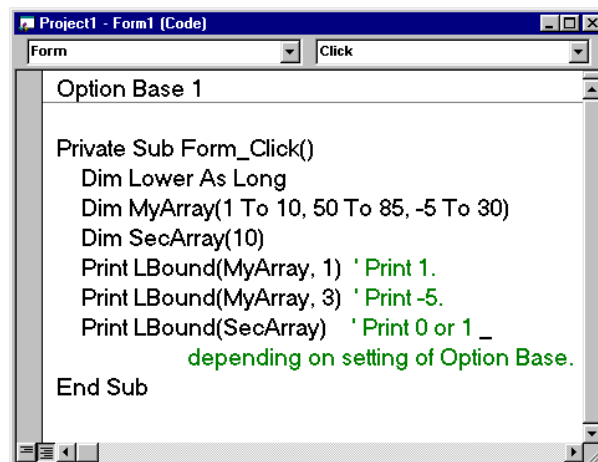
```
Dim Upper As Long
Dim MyArray(1 To 10, 50 To 85, 10 To 30)
Dim SecArray(10)
Upper = UBound(MyArray, 1)     'Returns 10.
Upper = UBound(MyArray, 3)     'Returns 30.
Upper = UBound(SecArray)       'Returns 10.
```

שים לב שהארגומנט השני, המייצג את המימד במערך, הוא רשות וכאשר המערך הוא חד-מימדי ניתן לוותר עליו. כאשר המערך הוא בעל מספר מימדים והארגומנט המציין את המימד הנבדק במערך לא נשלח לפונקציה, הפונקציה תתייחס למימד הראשון בלבד.

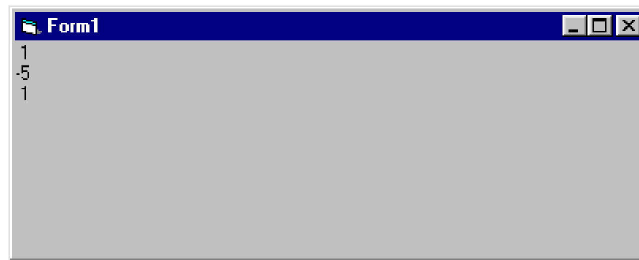
הפונקציה Lbound

בדומה לפונקציה Ubound, הבודקת את הגבול העליון, הפונקציה Lbound בודקת את הגבול התחתון של המערך. הפונקציה מקבלת שני פרמטרים, את שם המערך ואת המימד הנבדק, ומחזירה ערך Long המייצג את הגבול התחתון של המערך.

התרשימים 7.7 ו- 7.8 מדגימים שימוש בפונקציה Lbound, במבט קוד ובמבט ריצה בהתאמה.



תרשים 7.7



תרשים 7.8

כדי להימנע משגיאת חריגה מגבולות המערך, מומלץ לבדוק תחילה את טווח האינדקסים שלו לפני ביצוע פקודת השמה. נבצע זאת באמצעות שתי הפונקציות Ubound ו-Lbound.

מערכים דינמיים

במערך קבוע הגדרנו מראש, בשורת ההגדרה, את **גודל המערך**. לפעמים נרצה להשתמש במערך שגודלו אינו ידוע מראש בעת כתיבת התוכנית. ניקח לדוגמה, מזכירה באוניברסיטה הרושמת את פרטי המועמדים להתקבל לפקולטה מסוימת. אם נצהיר על מערך בגודל 100 איברים, אשר יקלוט את נתוני המועמדים, פירוש הדבר שאסור לקבל יותר מ-100 מועמדים. כאשר נרצה להכניס את המועמד ה-101, התוכנית תפסיק לפעול ותודיע על שגיאת חריגה מתחום המערך.

כדי להתגבר על בעיה זו קיים **המערך הדינמי** (Dynamic Array). מערך זה מאפשר לשנות את גודלו בזמן פעולת התוכנית ולהוסיף לו איברים כפי שדרוש. במקרה זה אין צורך להגדיר מראש את מספר האיברים במערך, כמו במערך קבוע. בכל פעם שנזדקק לאיבר נוסף, נוסיף אותו באופן דינמי למערך. אפשר להגדיל את המערך בשתי דרכים: האחת, ליצור מערך חדש שונה בגודלו ללא שמירת הערכים הקודמים; השנייה, ליצור מערך חדש אולם לשמור בו את הערכים הקודמים.

הגדלת המערך ללא שמירה על ערכיו הקודמים

כמו במערך רגיל, עלינו להצהיר על המערך קודם שנשתמש בו, אך במקרה זה נשאיר ריקים את הסוגריים, אשר בתוכם נרשם הטווח. לדוגמה,

```
Dim DynArray()
Dim Students()
Dim Grades() As Integer
```

כדי להגדיל את המערך להוספת איבר, נשתמש במשפט **ReDim**. לדוגמה,

```
ReDim DynArray(1)
ReDim Students(1)
ReDim Grades(1)
```

בדוגמה זו הגדלנו את המערך ועתה הוא בעל שני איברים (1 ו- 0). להוספת איבר נבצע פעולה דומה:

```
Redim DynArray(2)
ReDim Students(2)
ReDim Grades(2)
```

עכשיו, כל מערך מכיל שלושה איברים (0-2).

ניתן כמובן, להשתמש במונה להגדרת מספר האיברים, לדוגמה:

```
Dim Counter As Integer
Dim DynArray() As Integer
Counter = 1
ReDim DynArray(counter)
```

להוספת איבר, נכתוב כך:

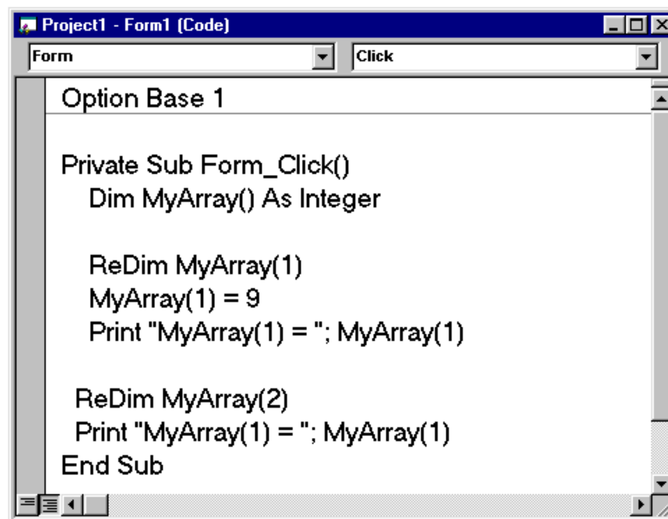
```
ReDim DynArray(counter + 1)
```

או כך:

```
counter = counter + 1
ReDim DynArray(counter)
```

הגדלת המערך עם שמירה על ערכיו הקודמים

בכל פעל שמגדילים את מספר איברי המערך ומוסיפים לו איבר, נוצר למעשה מערך חדש. המשפט **ReDim** גורם אמנם להגדלת המערך, אך יוצר מערך חדש ריק (עם איברים מסוג variant), או אפס (במערך מסוג משתנה מספרי), או עם מחרוזת ריקה (במערך מסוג String). כל הנתונים שהיו במערך הקודם **אובדים** (תרשימים 7.9 ו- 7.10).



תרשים 7.9

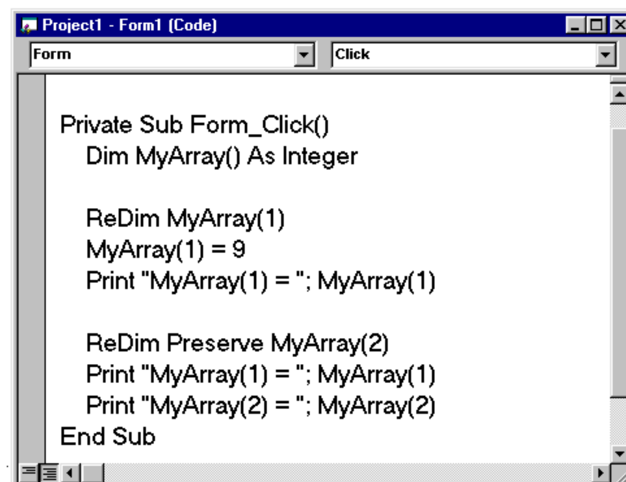


תרשים 7.10

כדי לשמור על הערכים הקודמים, נוסיף להוראה ReDim את המילה השמורה **Preserve**. לדוגמה,

```
ReDim Preserve Students(Counter + 1)
```

שורת קוד זו מורה לתוכנית להגדיל את המערך, אך לשמור על ערכיו הקודמים (תרשימים 7.11 ו- 7.12).



תרשים 7.11



תרשים 7.12

כבמערך רגיל, כך גם במערך דינמי ניתן להגדיר את הגבול העליון והתחתון.

לדוגמה,

```
Dim DynArray() As Integer
ReDim DynArray(10 To 20)
```

שים לב שניתן לשנות אך ורק את טווח המערך, אך לא ניתן לשנות את מספר המימדים שלו! כדי לא לחרוג בטעות מהגבול העליון של המערך, ניתן להשתמש בפונקציה Ubound כאשר מגדילים את המערך הדינמי. לדוגמה,

```
Dim DynArray() As Integer
ReDim Preserve DynArray(1)
ReDim Preserve DynArray(UBound(DynArray) + 1)
```

תרגול

1. הצהר על מערך קבוע בגודל 10, קלוט לתוכו 10 ציונים והדפס את הסכום ואת הממוצע שלהם.
2. בצע את תרגיל 1, אך הפעם בצע זאת בעזרת מערך דינמי השומר על ערכיו בכל פעם שהוא גדל.

8: מבני קבלת החלטה

פעמים רבות עומדת התוכנית מול שאלה או תנאי, כאשר התשובה לשאלה היא כן/לא, או האם התנאי מתקיים (True) או שאינו מתקיים (False). בהתאם לתשובה, או לקיום התנאי, נקבעת ההחלטה להמשך התוכנית.

את היכולת לשאול שאלה ולקבל החלטה על פי התשובה המתקבלת, נותנת ויז'ואל בייסיק בעזרת מבני ההחלטה הבאים:

```
1.If..Then
2.If..Then..Else
3.Select Case
```

המבנה If..Then

המבנה If..Then מאפשר למתכנת לבצע משפט מותנה, או משפטים מותנים על פי תוצאת הבדיקה של התנאי. המשפטים שיבוצעו מותנים בשאלה, או בתנאי אשר התשובה עליהם היא כן/לא, או האם התנאי מתקיים או לא. אם התשובה לשאלה היא כן, או שתנאי מתקיים, התוכנית מבצעת את המשפט או המשפטים שנכתבו על ידי המתכנת בהמשך המבנה, לאחר מילת המפתח Then; אם התנאי אינו מתקיים, התוכנית מדלגת על משפטים אלה וממשיכה במשפט הבא.

תחביר הפקודה לביצוע משפט אחד כאשר התנאי מתמלא, או שמתקבלת התשובה **כן**:

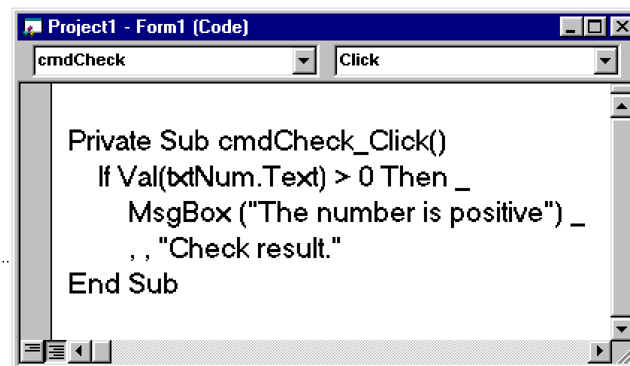
```
If condition Then Statement
```

לדוגמה, עיין בטופס המוצג בתרשים 8.1:

תרשים 8.1

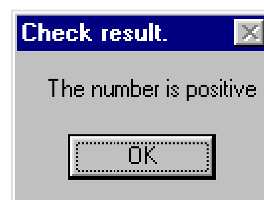
בטופס מוצגת תיבת טקסט שלתוכה המשתמש מתבקש להקיש מספר כלשהו. מתחת נמצא הלחצן Check, אשר בלחיצה עליו התוכנית בודקת אם המספר שהוקלד על ידי המשתמש חיובי (גדול מאפס). אם המספר שהקליד המשתמש גדול מאפס, תוצג תיבת הודעה מתאימה.

את הבדיקה אם המספר חיובי או לא, נערוך בעזרת מבנה ההחלטה **If..Then**. מבנה קבלת ההחלטה כולל תנאי או שאלה, אשר התשובה עליהם היא כן או לא. במקרה שלנו, השאלה היא האם המספר שהמשתמש הקליד הוא חיובי, או במילים אחרות, האם הוא גדול מאפס? תרשים 8.2 מתאר את שורות הקוד הבוחנות את התנאי.



תרשים 8.2

המאפיין Text של הפקד txtNum מחזיק בערך שהמשתמש הקליד בתיבת הטקסט. מכיון שהערך שמור כ-String, דרושה הפונקציה Val() כדי להמירו למספר. בעזרת האופרטור "גדול" (>), הוצג התנאי במבנה ההחלטה If. אם המספר גדול מאפס, התנאי מתקיים ומוצגת תיבת ההודעה MsgBox (תרשים 8.3). כאשר התנאי לא מתקיים, התוכנית מדלגת על מבנה ההחלטה וממשיכה הלאה. בדוגמה שלנו, התוכנית מדלגת אל שורת הקוד End Sub והשיגרה מסתיימת.



תרשים 8.3

בתוך מבנה ההחלטה אפשר להכיל **יותר** ממשפט פעולה אחד, אך במקרה זה התחביר של משפט If שונה במקצת:

```
If condition Then
   statments
End If
```

שים לב, שכאשר נכתבים מספר משפטים לביצוע בתוך מבנה ההחלטה, יש להוסיף את המשפט "End If" בסוף מבנה ההחלטה. משפט זו מודיע לתוכנית היכן מסתיימים המשפטים המותנים הנכללים במבנה ההחלטה. גם כאן, אם התנאי מתקיים המשפטים מתבצעים, ואם לאו - התוכנית ממשיכה כרגיל ללא ביצוע משפטים אלה. לדוגמה:

```
Dim Grade As Integer
If Grade >= 90 then
   Grade = Grade + 5     'Bonus 5 points
   Print "Very good!"
End if
```

התנאי במבנה ההחלטה משווה בדרך כלל שני משתנים או משתנה וקבוע. לדוגמה:

```
Dim Grade As Integer
If Grade >= 90 then Print "Very good!"
```

או:

```
Dim Grade As Integer
If Grade >= HighGrade then Print "Excellent!"
```

ניתן גם להשוות בין שני ביטויים מורכבים, כמו לדוגמה:

```
If (Psychometric + BagTest) / 2 > 380 Then Pass = True
```

משפט תנאי עם משתנה (אופרנד) אחד

כאשר במשפט התנאי נמצא משתנה מספרי בודד, ההתייחסות לתנאי תהיה בצורה הבאה: אם הערך המספרי של המשתנה הוא 0, התנאי אינו מתקיים והמשפטים הבאים אחריו אינם מתבצעים. אם ערך המשתנה שונה מאפס (וללא תלות בערכו הממשי), התנאי מתקיים והמשפטים המותנים במבנה ההחלטה מתבצעים. לדוגמה:

```
Dim Pass As Integer
Pass = 5
If Pass Then
   MsgBox ("The condition is True")
End If

Dim Pass As Integer
Pass = 0
If Not Pass Then
   MsgBox ("The condition is True")
End If
```

במבנה ההחלטה הראשון נבדק המשתנה היחיד Pass, ולכן התנאי מתקיים כל עוד ערכו שונה מאפס. בדוגמה זו ערכו של Pass הוא 5, ולכן התנאי מתקיים והתוכנית מציגה תיבת הודעה שבה "The condition is True". לעומת זאת, במבנה ההחלטה השני ערך המשתנה Pass הוא 0. הפעם הבדיקה היא "Not Pass", ובמילים אחרות,

הפיכת משמעות התשובה הנכונה. מכיון שערך Pass הוא 0, זהו ערך שקרי, שהפיכתו על ידי Not תתן לנו תשובת "אמת"; לכן, גם במקרה זה התנאי יתקיים והשורה "The condition is True" תוצג בתיבת ההודעה.

בדוגמאות אלו כתבנו את המשפט End If למרות שיש משפט אחד בלבד. הסיבה לכך היא, שכאשר המשפט נכתב באותה שורה בה כתוב התנאי, בהמשך למילה Then, אין צורך ב- End If, אך כאשר המשפט נכתב בשורה נפרדת, יש לסיים את מבנה ההחלטה ב- End If, כדי למנוע אי הבנות הן מצד המתכנת והן מצד מהדר התוכנה.

במבנה ההחלטה If..Then מבצעת התוכנית את המשפטים המותנים רק כאשר התנאי מתקיים. אם התנאי אינו מתקיים התוכנית מדלגת על משפטים אלה, וממשיכה הלאה, במשפט העוקב.

לפעמים נרצה לבצע משפטים מותנים גם כאשר התנאי אינו מתקיים. למשל, לטופס שמוצג בתרשים 8.1, נוסיף את האפשרות לתת למשתמש הודעה על מספר שלילי במקרה שהמספר שהקיש קטן מאפס. לצורך זה נשתמש במבנה ההחלטה If..Then..Else.

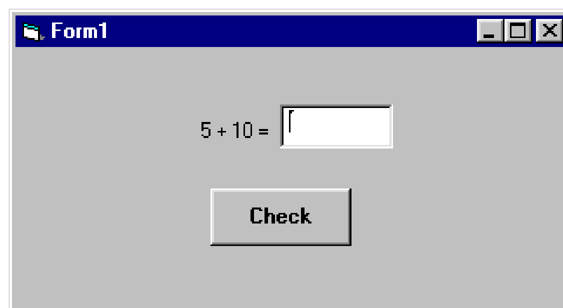
המבנה If..Then..Else

מבנה ההחלטה If..Then..Else מאפשר להפריד בין קיום התנאי לבין אי קיומו, ולבצע פעולה שונה, או פעולות שונות, והכל - במסגרת משפט If. אם התנאי מתקיים מתבצעים המשפטים שלאחר מילת המפתח **Then**; אחרת (כלומר, **Else**) יתבצעו משפטים אחרים. במבנה ההחלטה If..Then אם התנאי אינו מתקיים התוכנית ממשיכה למשפט העוקב. במבנה ההחלטה If..Then..Else מתבצעים משפטים מסוימים במצב שבו התנאי אינו מתקיים והם רשומים לאחר מילת המפתח Else. רק אחר כך התוכנית ממשיכה למשפט העוקב.

תחביר ההחלטה הוא:

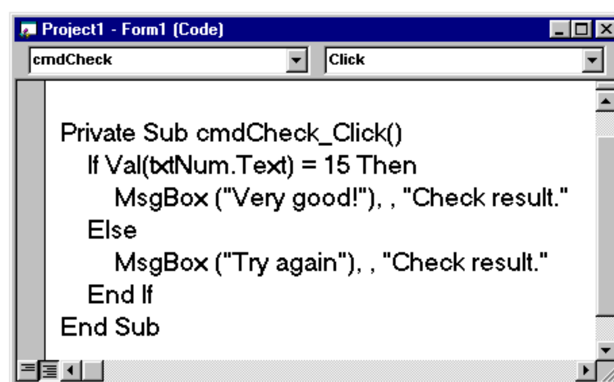
```
If condition Then
   True part statements
Else
   False part statements
End If
```

לדוגמה, על המשתמש לענות על שאלת חשבון המוצגת בפניו (ראה תרשים 8.4). אם הוא ענה נכון, התוכנית תציג לפניו הודעה מתאימה. אם הוא שגה, התוכנית תציג בפניו הודעת שגיאה.



תרשים 8.4

במקרה זה עלינו להציג הודעה גם כאשר התנאי מתקיים, וגם כשאינו מתקיים. על כן יש לכתוב משפטים מתאימים לשני המצבים: למצב בו התנאי מתקיים וגם למצב בו התנאי אינו מתקיים. לצורך זה נשתמש במבנה ההחלטה If..Then..Else (תרשים 8.5).

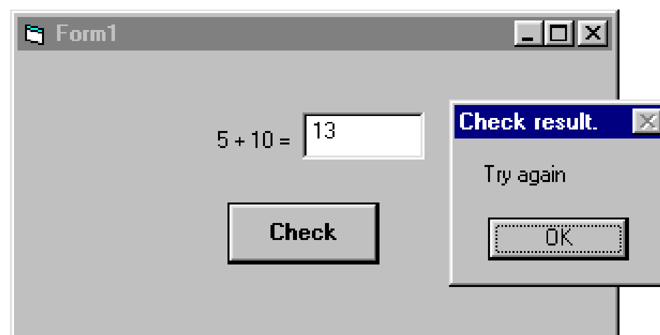


תרשים 8.5

התנאי העומד לבדיקה הוא:

```
Val(txtNum.Text) = 15
```

כאשר התנאי מתקיים, או במילים אחרות כאשר המשתמש ענה תשובה נכונה, מוצגת הודעה מתאימה. אם המשתמש שגה והתנאי לא מתקיים, התוכנית לא תדלג על מבנה ההחלטה ותמשיך בפעולתה. במקרה זה היא תדלג אל המשפט או המשפטים שלאחר מילת המפתח **Else** שבמבנה ההחלטה, ותבצע את הכתוב בו. בדוגמה שלנו התוכנית תציג הודעת שגיאה (תרשים 8.6).



תרשים 8.6

היררכיית ElseIf

נבחן את שורות הקוד הבאות:

```
Dim Num As Integer
Num = 1

If Num = 1 Then
   MsgBox "The number is one."
End If

If Num = 2 Then
   MsgBox "The number is two."
End If

If Num = 3 Then
   MsgBox "The number is three."
End If
```

שלושת מבני ההחלטה בודקים את ערך המשתנה Num. בדוגמה, ערך Num הוא 1, ולכן התנאי הראשון מתקיים והמשפט הבא אחריו מתבצע.

לאחר ביצוע המשפט וסיום מבנה ההחלטה הראשון, פונה התוכנית למבנה ההחלטה השני ובודקת את התנאי שלו. התנאי אינו מתקיים והתוכנית ממשיכה למבנה ההחלטה הבא. גם כאן נבחן התנאי והוא אינו מתקיים ולכן התוכנית ממשיכה.

מצב שבו התוכנית חוזרת על הבדיקה אינו תכנות נכון והתוכנית אינה יעילה. אם מתברר שערך Num הוא 1, מיותר לבדוק את מבני ההחלטה הבאים, כי הם יתנו את אותה תוצאה, ולא יתקיימו.

כדי לייעל את הקוד נשתמש בהיררכיית **ElseIf**. נבחן את שורות הקוד האלו, שבהן אנו בודקים אם ערכו של Num הוא 1, 2, או 3 ופועלים בהתאם:

```
Dim Num As Integer
Num = 1

If Num = 1 Then
   MsgBox "The number is one."
ElseIf Num = 2 Then
   MsgBox "The number is two."
ElseIf Num = 3 Then
   MsgBox "The number is three."
End If
```

בשורות אלו, מכיון ש-Num=1, התנאי הראשון מתקיים והתוכנית מבצעת את המשפט:

```
MsgBox "The number is one."
```

אך הפעם, שני מבני ההחלטה הבאים שייכים לחלק ה- Else אליו פונה התוכנית רק כאשר התנאי לא התקיים. בדוגמה שלנו, אם כן, התוכנית תדלג על חלק זה ותמשיך הלאה. בסופו של דבר התוצאה בשתי הדוגמאות תהיה דומה ונקבל את אותה תיבת הודעה. אך יעילות ומהירות ביצוע התוכנית גדולה יותר בדוגמה השנייה, בה נבדק התנאי הראשון בלבד.

המבנה Select Case

עיין שוב בשורות הקוד האלו:

```
Dim Num As Integer
Num = 1

If Num = 1 Then
   MsgBox "The number is one."
ElseIf Num = 2 Then
   MsgBox "The number is two."
ElseIf Num = 3 Then
   MsgBox "The number is three."
End If
```

בדקנו בהם אם ערכו של Num הוא 1, 2, או 3. נניח שברצוננו להציג הודעה דומה, כאשר הערכים של Num הם 1 או 2, ולהציג הודעה שונה כאשר הערך הוא 3. השימוש במבנה ההחלטה If..Then..Else במקרה זה מסבך מעט את שורות הקוד, והן אינן קריאות וברורות. לדוגמה,

```
Dim Num As Integer
Num = 1

If Num = 1 Or Num = 2 Then
   MsgBox "The number is one or two."
ElseIf Num = 3 Then
   MsgBox "The number is three."
End If
```

בדוגמה זו שורות הקוד אינן מורכבות כי נבדקים בהן ערכים מעטים. אך מה יקרה אם נרצה שבמשפט If נרצה לבדוק 50 ערכים למשל. כדי לבדוק ערכים שונים נכתוב שורת קוד כזו, למשל:

```
If Num = 1 Or Num = 10 Or Num = 35 ... Or Num = 78 Then
```

שורות הקוד שלנו הפכו למסורבלות ובלתי קריאות. על כן, נעדיף במקרה כזה את מבנה ההחלטה **Select Case**, שהתחביר שלו הוא:

```
Select Case variable
   Case value
      statements
   Case value
      statements
End Select
```

כאן, Variable הוא הערך שנבדק ו- Value הוא הערך שלעומתו בודקים את Variable. Statements הם הפקודות שתתבצענה בכל מקרה שהתוצאה היא חיובית ("כן"). לדוגמה:

```
Dim Num As Integer
Num = 2

Select Case Num
   Case 1
      MsgBox "The number is one."
   Case 2
      MsgBox "The number is two."
   Case 3
      MsgBox "The number is three."
End Select
```

בדוגמה זו נבחן ערכו של משתנה בודד Num. במקומו אפשר לכתוב גם ביטוי מורכב. כמו בדוגמה הקודמת גם כאן נבדוק אם ערכו הוא 1, 2, או 3. לדוגמה,

```
Dim Num As Integer
Num = 6

Select Case (Num * 2) / 4
   Case 1
      MsgBox "The number is one."
   Case 2
      MsgBox "The number is two."
   Case 3
      MsgBox "The number is three."
End Select
```

אלו הן דוגמאות פשוטות הדומות מאוד למבנה ההחלטה If..Then..Else. באמצעות המשפט Select Case אפשר לבצע משפט אחד על **טווח** ערכים של המשתנה הנבחן. לדוגמה,

```
Dim Num As Integer
Num = 5

Select Case Num
   Case 1
      MsgBox "The number is one."
   Case 2 To 6
      MsgBox "The number is between two and six."
   Case 10
      MsgBox "The number is ten."
End Select
```

אם לדוגמה, ערך המשתנה Num הוא 3, מבנה ההחלטה יפנה לחלק המטפל בערך זה. במקרה שלנו, 3 נמצא בטווח שבין 2 ל- 6, ולכן יבוצע המשפט:

```
MsgBox "The number is between two and six."
```

באופן דומה מתבצע משפט אחד על **ערכים שונים** של המשתנה. לדוגמה,

```
Dim Num As Integer
Num = 3

Select Case Num
   Case 1
      MsgBox "The number is one."
   Case 3,6
      MsgBox "The number is tree or six."
   Case 10
      MsgBox "The number is ten."
End Select
```

בדומה לחלק "Else" במבנה ההחלטה If..Then..Else קיים גם במבנה ההחלטה Select Case חלק ששמו **Case Else**, שבו נרשמים המשפטים לביצוע כאשר אף אחד מהתנאים שנבדקו אינו מתקיים. כדוגמה, ניצור את הטופס המוצג בתרשים 8.7.

תרשים 8.7

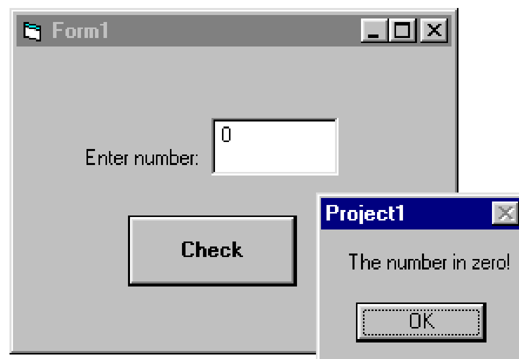
בטופס זה המשתמש מתבקש להקליד ערך בתיבת הטקסט וללחוץ על לחצן Check. בתגובה תופיע תיבת הודעה המציינת אם המספר שהוקלד הוא חיובי, שלילי או אפס.

לביצוע מטלה זו נכתוב את שורות הקוד האלו עבור האירוע Click של הלחצן cmdCheck (תרשים 8.8).

```
Private Sub cmdCheck_Click()
  Select Case Val(txtNum.Text)
    Case Is > 0
      MsgBox "The number in positive!"
    Case Is < 0
      MsgBox "The number in negative!"
    Case Else
      MsgBox "The number in zero!"
  End Select
End Sub
```

תרשים 8.8

שים לב שניתן להשתמש גם באופרטורים "גדול" (>) ו- "קטן" (<) בתוספת המילה השמורה "Is". ואם ערך המשתנה או הביטוי אינו בטווח הערכים של שני חלקי משפט Case, התוכנית לא תדלג על מבנה ההחלטה, אלא תפנה לחלק Case Else של המבנה (תרשים 8.9).



תרשים 8.9

תרגול

1. תן למחשב לבחור מספר באופן אקראי (היעזר בפונקציה Rnd) ואפשר למשתמש לנחש מהו המספר שנבחר. אם המספר של המשתמש גדול מהמספר שנבחר על ידי המחשב, או קטן ממנו, הצג הודעה מתאימה. המשחק יסתיים כאשר המשתמש ינחש את המספר.
2. הצג טופס עם תפריט ומחירים. אפשר למשתמש להזמין כרצונו ובסוף ההזמנה הצג את מחירה הכולל של ההזמנה.

9: לולאות

פעמים רבות על המתכנת להפעיל קטע קוד כלשהו פעמים אחדות. במקום לרשום משפט אחד או כמה משפטים בכל פעם מחדש, כמספר הפעמים הדרוש, אפשר להפעיל את מבנה הלולאה. **לולאה (Loop)** כוללת משפט אחד או מספר משפטים, המבוצעים מספר מוגדר של פעמים או באופן מחזורי, עד אשר מתקיים תנאי מסוים, או שהוא אינו מתקיים. נניח שהמתכנת רוצה להציג בטופס את המספרים 1-100. במקום לכתוב 100 משפטים במאה שורות קוד, יעיל ופשוט יותר לכתוב משפט הדפסה אחד אשר יחזור על עצמו, בעזרת הלולאה, 100 פעמים.

הלולאה Do..While

הלולאה Do..While (״**עשה כל עוד**״) מבצעת משפט אחד או מספר משפטים באופן חוזר ונשנה מספר רב של פעמים. הלולאה כוללת תנאי לביצוע הלולאה. במילים אחרות, כל עוד התנאי מתקיים (True) הלולאה ממשיכה והמשפטים שבתוכה מתבצעים. ברגע שהתנאי מפסיק להתקיים, הלולאה מפסיקה והתוכנית ממשיכה במשפט שלאחר משפט הלולאה. נקודת העצירה של הלולאה היא כאשר תנאי ביצוע הלולאה מקבל ערך False (אינו מתקיים).

תחביר הלולאה Do..While הוא:

```
Do While condition
   statements
Loop
```

לדוגמה:

```
Dim Again As Integer
Again = 1
Do While Again <= 4
   Print Again
   Again = Again + 1
Loop
```

מתי נשתמש בלולאה Do..While?

בלולאה זו נשתמש כאשר מספר מחזורי הלולאה אינו ידוע מראש, והדבר היחיד שידוע הוא התנאי לביצוע הלולאה, או באיזה מקרים הלולאה תמשיך להתבצע. תנאי זה הוא **תנאי ביצוע הלולאה**, והלולאה תהיה במבנה Do..While.

הלולאה Do..Loop

סגנון שונה של אותה לולאה הוא המבנה Do..Loop, כפי שנראה להלן:

```
Do
   statements
Loop While condition
```

לדוגמה:

```
Dim Again As Integer
Again = 1
Do
   Print Again
   Again = Again + 1
Loop While Again <= 4
```

גם במבנה זה ביצוע הלולאה תלוי בקיום תנאי. כל עוד התנאי מתקיים הלולאה מתבצעת, וכאשר התנאי אינו מתקיים הלולאה מפסיקה להתבצע.

ההבדל בין שני הסגנונות הוא, שבלולאה Do..While התוכנית בודקת תחילה את התנאי ורק אם הוא מתקיים המשפטים בתוך הלולאה מתבצעים. אם התנאי אינו מתקיים בכניסה הראשונה ללולאה, המשפטים שבתוכה לא יתבצעו אפילו פעם אחת.

לעומת זאת, בלולאה Do..Loop מתבצעים תחילה המשפטים שבתוך הלולאה, ורק אחר כך נבדק התנאי. גם אם התנאי אינו מתקיים כלל, המשפטים שבתוך הלולאה יבוצעו פעם אחת לפחות.

הלולאה Do..Until

לולאה זו דומה מאוד ללולאת Do..While בצורת פעולתה, וגם לה יש שני סגנונות. הלולאה מבצעת משפט או מספר משפטים, מספר פעמים התלוי בתנאי הלולאה. אך שונה מהלולאה Do..While, הלולאה **Do..Until** מבצעת את המשפטים בלולאה כל עוד התנאי **אינו** מתקיים. בלולאה זו, תנאי הלולאה הינו תנאי **סיום** והלולאה תמשיך בפעולתה כל עוד התנאי אינו מתקיים.

מבנה הלולאה הוא:

```
Do Until condition
   statements
Loop
```

במבנה זה, אם התנאי מתקיים – הלולאה אינה מתבצעת כלל.

במבנה הבא, לדוגמה, הלולאה תתבצע לפחות פעם אחת:

```
Do
   statements
Loop Until condition
```

הלולאה For..Next

הלולאה For..Next, כמו הלולאה Do..While, מבצעת מספר פעמים משפט אחד או משפטים. ראינו שבלולאה Do..While מספר מחזורי הלולאה אינו ידוע, אך בלולאה For..Next מספר מחזורי הלולאה מוגדר מראש. במילים אחרות, אם ידוע מראש כמה פעמים תבצע הלולאה את המשפטים הכתובים בה, נעדיף להשתמש בלולאה For..Next.

תחביר הלולאה:

```
For counter = start To end [Step increment]
   statements
Next counter
```

לדוגמה:

```
For I = 1 to 10 step 2
   print "Hello"
next I
```

בלולאה For..Next נמצא משתנה, המשמש **מונה לולאה** (Loop counter). מונה זה עוקב אחר מספר הפעמים שהלולאה מתבצעת. המונה הוא משתנה מספרי כלשהו, ואינו מוגדר בהכרח בשם counter.

הארגומנט **start** בלולאה קובע את ערכו ההתחלתי של המונה והארגומנט **end** קובע את ערכו המקסימלי. בדוגמה הקודמת, המשתנה `I` הוא מונה הלולאה, ערכו ההתחלתי הוא 1 וערכו המקסימלי (הסופי) הוא 10.

הארגומנט **increment** קובע את ערך הקידום של מונה הלולאה. בדוגמה הקודמת ערכו הוא 2, ולכן הוא גדל בקפיצות של 2. לפיכך, המונה מתחיל ב-1 מתקדם ל- 3, ל- 5, 7 ו-9 והלולאה מסתיימת, כי הערך הבא הוא 11 אשר גדול מהערך המקסימלי של המונה. הלולאה מתבצעת לפי תנאי זה 5 פעמים בלבד.

כאשר ערך הקידום של המונה הוא 1, אין צורך לציין את הארגומנט increment, כי זו ברירת המחדל. הלולאות האלו זהות לחלוטין, ובשתיהן המילה "Hello" תודפס 10 פעמים:

```
For I = 1 to 10 step 1
   print "Hello"
Next I

For J = 1 to 10
   print "Hello"
Next J
```

כדי ללמוד על שימוש פשוט בלולאה מסוג For..Next, נכתוב את המילה "GOOD" באיברים 3 עד 5 של מערך Table. לצורך זה דרושות פקודות אלו:

```
For I = 3 to 5
   Table(I) = "GOOD"
next I
```

הארגומנט increment יכול לקבל גם ערכים חיוביים וגם ערכים שליליים. כאשר ערך הארגומנט increment חיובי, הערך ההתחלתי של המונה **קטן** מערכו של הערך העליון שלו; כאשר ערך הארגומנט increment שלילי, הערך ההתחלתי של המונה גדול מערכו הסופי (במקרה זה המינימלי). לדוגמה,

```
For I = 10 To 1 Step -1
   Print I
Next I
```

בלולאה זו יודפסו המספרים 1 עד 10 בסדר יורד! האם ברור מדוע?

אם הערך ההתחלתי של המונה קטן מערכו המקסימלי וערך הארגומנט increment שלילי - הלולאה **לא** תתבצע כלל.

הבדל נוסף שיש בין הלולאה For..Next לבין הלולאה Do..Loop הוא ניצול ערך מונה הלולאה. התבונן בשתי הלולאות הבאות:

```
For I = 1 To 10
   Print "Hello"
Next I

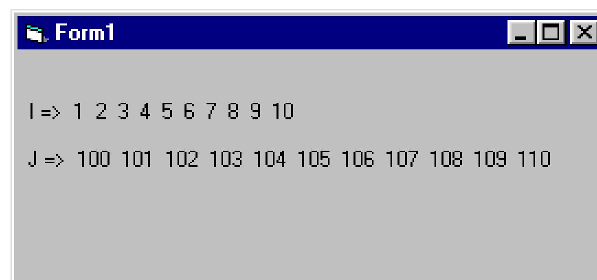
For J = 100 To 110
   Print "Hello"
Next J
```

בשני המקרים תודפס המילה "Hello" 10 פעמים. במקרה זה אין כל הבדל בין שתי הלולאות. אך הנה דוגמה שונה עבור שתי הלולאות:

```
For I = 1 To 10
   Print I
Next I

For J = 100 To 110
   Print J
Next J
```

במקרה זה יש הבדל גדול בין שתי הלולאות, כי בלולאה הראשונה יודפסו הערכים המספריים 1 עד 10, ובלולאה השנייה יודפסו הערכים המספריים 100 עד 110 (תרשים 9.1). הבדל זה נובע מהעובדה שייחסנו משמעות **לערך** של מונה הלולאה.



תרשים 9.1

כאשר מונה הלולאה משמש כמונה בלבד, אין כל חשיבות לערכי ההתחלה והסיום, כל עוד הטווח נשמר,ובדוגמאות הקודמות הטווח היה 10, לביצוע 10 מחזורי לולאה. אך כאשר ערך המונה חשוב, יש הבדל רב אם ערכו נע בטווח שבין 1 ל- 10 או בין 100 ל- 110.

לולאות מקוננות

משפטים הכתובים בתוך הלולאה מבוצעים מספר פעמים, על פי מונה הלולאה, או על פי התנאי לסיום הלולאה. משפטים אלה יכולים להיות כל משפט חוקי בשפת ויז׳ואל בייסיק, כמו פקודת השמה, מבנה קבלת החלטה ואפילו לולאה. כלומר, ניתן להגדיר לולאה בתוך לולאה. במקרה זה נאמר כי אלו הן **לולאות מקוננות** (Nested loops).

לדוגמה:

```
For I = 1 To 10
   For J = 1 To 10
      Print "Hello"
   Next J
Next I
```

שים לב, שמילת המפתח **Next** המסיימת את הלולאה הפנימית, מוצבת **לפני** הסיומת **Next** של הלולאה החיצונית.

כמה פעמים תודפס לדעתך המילה "Hello"? אם אמרת 100 - צדקת, והסיבה לכך היא שבכל מחזור של הלולאה החיצונית, הלולאה הפנימית מבצעת את **כל** המחזורים המוגדרים לה. בדוגמה זו, בכל מחזור של הלולאה החיצונית (בבקרת המשתנה **I**), הלולאה הפנימית (בבקרת המשתנה **J**) מבצעת 10 מחזורים. מכיון שהלולאה החיצונית מבצעת 10 מחזורים גם היא, מתבצעים בסך הכל 100 (10X10) מחזורים של הלולאה **הפנימית**.

הבה נתבונן בלולאה זו:

```
For I = 1 To 10
   For J = 1 To 10
      Print "Hello"
   Next J
   Print "world"
Next I
```

המילה "Hello" תודפס באמצעות לולאה זו 100 פעמים, מכיון שההדפסה נעשית בלולאה הפנימית; על כן, בכל מחזור של הלולאה החיצונית מתבצעים כל מחזורי הלולאה הפנימית. לעומת זאת, המילה "world" תודפס 10 פעמים בלבד, מכיון שהמשפט Print "world" הוא חלק מהלולאה החיצונית (ולא הפנימית), ולכן הוא יתבצע כמספר מחזורי הלולאה החיצונית, כלומר - 10 פעמים.

גם בלולאות מקוננות ניתן לנצל את ערכי המונים. הנה דוגמה של לולאות מקוננות המדפיסות את לוח הכפל בעזרת שימוש בערכי מוני הלולאה.

```
For I = 1 To 10
   For J = 1 To 10
      Print I * J;
   Next J
   Print chr(13)    ' New line
Next I
```

ניתן להשתמש בערך המונה גם לאחר סיום הלולאה ויציאה ממנה. המונה הוא משתנה לכל דבר, אשר שומר על ערכו בהתאם לאורך החיים שלו. אם זהו משתנה מקומי, אורך חייו עד סוף השיגרה או הפונקציה שבה הוגדר, ואם הוא משתנה מודולרי, הוא שומר על ערכו במשך חיי המודול או הטופס.

שימוש בערך המשתנה לאחר סיום הלולאה מחייב תשומת לב רבה. מבני לולאה שונים בודקים באופן שונה את תנאי הסיום, ולכן יש לבדוק תחילה איך הלולאה נוהגת. כמו כן, בלולאה מסוג For ישנו ערך הגדלה שצריך להתחשב בו. הכלל אומר שערך המונה בסיום הלולאה יהיה כערכו המקסימלי של הטווח + ערך ההגדלה (אם ערכו האחרון היה שווה לערך המקסימלי). לדוגמה:

```
For I = 1 To 9 Step 2
   Print I
Next I
```

הלולאה תתבצע עם הערכים 1, 3, 5, 7, 9, תגדל ב-2 כמו בכל מחזור של הלולאה, וכאשר ערך המונה יהיה 11 הלולאה תסתיים והתוכנית תמשיך למשפט הבא.

דוגמה כאשר ערך המונה אינו מקבל את הערך המקסימלי של הטווח:

```
For I = 0 To 10 Step 3
   Print I
Next I
```

במקרה זה יודפסו הערכים הבאים: 0, 3, 6, 9, וכשמונה הלולאה יגיע ל-12 הלולאה תסתיים והתוכנית תמשיך במשפט הבא. במקרה זה, הערך האחרון של המונה היה 9 (ולא הערך המקסימלי), לכן ערך ההגדלה נוסף לערך האחרון של המונה, ולא לערכו המקסימלי. במקרה זה ערך המונה לאחר סיום הלולאה יהיה 12 (3+9=12).

הלולאה For Each..Next

לולאה זו דומה ללולאה For..Next. גם בלולאה זו מתבצע משפט, או שמתבצעים כמה משפטים, מספר מוגדר של פעמים. השוני בין הלולאות הוא, שבמקום מונה הקובע את מספר החזרות של הלולאה, אנו רואים שבלולאה **For Each..Next** הלולאה חוזרת על עצמה כמספר האיברים שבאוסף (collection) או במערך שהוגדר. היתרון בלולאה זו הוא בכך שניתן להפעיל אותה מבלי לדעת מראש כמה פעמים עליה להתבצע (בניגוד ללולאה For..Next בה צריך להגדיר במדויק את טווח הערכים של מונה הלולאה).

תחביר הלולאה הוא:

```
For Each element In group
   statements
Next element
```

בדוגמה הבאה מדפיסים את שמות כל הטבלאות שבמסד הנתונים Biblio.mdb:

```
Dim db As Database
Dim strFile As String
strFile = CurDir() & "\Biblio.mdb"
Set db = OpenDatabase(strFile)
For Each TableDef In db.TableDefs()
   Print Tabledef.Name
Next TableDef
```

שים לב, שכאשר הלולאה מחפשת במערך או באוסף, הארגומנט **element** חייב להיות מסוג **variant**. בלולאה זו אין אפשרות להשתמש במשתנה מסוג מבנה (User-defined Type).

יציאה מוקדמת מלולאה

לפעמים נרצה לצאת מהלולאה למרות שהיא לא סיימה עדיין את מספר החזרות הקצובות מראש. בדרך כלל יהיה תנאי כלשהו בתוך הלולאה, שעם קיומו (או אי-קיומו) הלולאה תפסיק לפעול, כמו למשל, זיהוי מצב חריג, או מצב שגיאה. יציאה ישירה מהלולאה נעשית באמצעות המשפט Exit. המשפט **Exit** תקף **בלולאה Do** וגם בלולאת **For**. לדוגמה שימוש בלולאה **For**, עם הפקודה **Exit For**:

```
For I = 1 To 10
   If I > 3 Then Exit For
Next I
```

באותו אופן נשתמש בלולאה **Do** עם הפקודה **Exit Do**. לדוגמה,

```
Do While I < 10
   If I > 3 Then Exit Do
   I = I + 1
Loop
```

שים לב, יציאה מלולאה באמצעות הפקודה Exit היא ישירה. הלולאה **אינה** ממשיכה לבצע את שורות הקוד המופיעות לאחר הפקודה הזו. התוכנית מדלגת לסוף הלולאה וממשיכה במשפט שאחריה. דוגמה זו מציגה שוב סיום של לולאה על פי תנאי:

```
For I = 1 To 10
   If I > 3 Then Exit For
   Print "Hello"
Next I
```

כל עוד ערכי המונה קטנים מ-3 הלולאה תדפיס את המילה "Hello". כאשר מונה הלולאה יקבל את הערך 4 והתנאי לא יתקיים, הלולאה תיפסק מייד והתוכנית לא

תבצע את פקודת ההדפסה "Print "Hello. כך, המילה "Hello" תודפס 3 פעמים בלבד (עבור הערכים 1, 2, 3 של המונה).

תרגול

1. תן למשתמש לבחור מספר, והצג הודעה אם המספר שהקיש הוא ראשוני או לא.
2. צור את הטופס הבא:

בלחיצה על Start הלחצנים Left ו-Right ינועו זה לקראת זה, וכאשר הם ייפגשו תוצג המילה "בום!" בתיבת הטקסט.

10: שגרות

שיגרה (subroutine) היא קטע קוד בעל שם. כאשר נרצה שהתוכנית תבצע את שורות הקוד שבתוך השיגרה, נקרא לשיגרה בשמה. התוכנית תפנה לשיגרה שזהו שמה ותבצע את שורות הקוד הכתובות בה.

מבנה השיגרה

הצהרה על שיגרה נעשית באופן הבא:

```
Sub subname()
   statements
End Sub
```

המילה **Sub** מסמלת את תחילת השיגרה, אחריה יבוא **שם** השיגרה **וצמוד** לו – **סוגריים**. בתוך השיגרה נכתבות שורות הקוד הרצויות, והשיגרה מסתיימת במילים **End Sub**.

השיגרה הבאה, מדפיסה את המשפט "Good Morning" בכל פעם שהיא נקראת:

```
Sub GoodMorning()
   Print "Good Morning"
End Sub
```

מעתה, בכל פעם שנרצה להדפיס את המשפט "Good Morning", פשוט נקרא לשיגרה והיא תדפיס את המשפט.

קריאה לשיגרה נעשית באמצעות ההוראה **Call**, הנה כך:

```
Call subname
```

לדוגמה:

```
Call GoodMorning
```

אחרי שהשיגרה מסיימת לבצע את שורות הקוד שבתוכה, התוכנית חוזרת אל שורת הקוד שקראה לשיגרה וממשיכה מיד אחריה. השיגרה יכולה להכיל בתוכה כל משפט חוקי בשפת ויז'ואל בייסיק, כמו למשל פקודות השמה, הצהרה על משתנים, לולאות, מבני החלטה ואף קריאה לשיגרה אחרת (או לעצמה במקרים מסוימים), ועוד.

תכנות מודולרי

תכנות באמצעות שגרות מחלק למעשה את שורות הקוד למספר קבוצות, או שגרות, במקום להציג אותן ברצף. תכנות זה נקרא בשם **תכנות מודולרי** (Modular programming). מתכנתים מקצועיים נוהגים לחלק את שורות הקוד לשגרות ולפונקציות, כאשר כל אחת מהן עוסקת ביחידת עיבוד מוגדרת ופשוטה.

מדוע לחלק את שורות הקוד באופן מודולרי, ולא לכתוב את כל התוכנית ברצף אחד ארוך? לשאלה זו יש תשובות רבות, והשתיים העיקריות הן:

1. שורות קוד המחולקות לשגרות ופונקציות הופכות את התוכנית לקריאה וברורה יותר. לאחר חודש מילואים אחד שלא נגענו בתוכנית ונצטרך לשחזר לעצמנו מה כתבנו (במקרה הטוב, ובמקרה הגרוע מה כתבו אחרים), לא נבין מה כתוב ב"מגילת" הקוד הארוכה הזו. לעומת זאת, כאשר התוכנית מחולקת לפונקציות ולשגרות שלכל אחת מהן תפקיד מוגדר משלה (בנוסף ל**תיעוד** קצר על כל שיגרה ופונקציה), קל ומהיר יותר יהיה להתמצא בשורות קוד אלו ולבצע בהן שינויים.
2. יעילות התכנות המודולרי נובעת מאופן המעקב אחרי ביצוע התוכנית. לעיתים (שאינן רחוקות) קורה שמתגלה "באג" בתוכנית. במקרה כזה, אם שורות הקוד כתובות ברצף אחד ארוך, יהיה קשה מאוד לעקוב אחר הטעות ולמצוא את מקור התקלה. לעומת זאת, כאשר התוכנית מחולקת לשגרות וכל שיגרה אחראית לפעולה מסוימת בתוכנית, קל יהיה בהתאם למיקום הטעות בתוכנית, להצביע על קטע הקוד (פונקציה, או שיגרה) אשר גרמו ל"באג". במקרה זה נתמקד בפתרון בעיה שמתגלית בשורות קוד של פונקציה או של שיגרה בלבד, ולא בכל שורות הקוד שבתוכנית.

פרמטרים וארגומנטים

נחזור ל"**שיגרה**". השיגרה מבצעת את שורות הקוד שבה בכל פעם שקוראים לה. שורות אלו קבועות, ובכל פעם שנקרא לפונקציה היא תבצע בדיוק את אותן שורות. אולם, השיגרה יכולה לקבל פרמטרים חיצוניים שיכולים להיות שונים בכל קריאה לשיגרה, ועל ידי כך לשנות את תוצאת השיגרה (שורות הקוד כמובן שאינן משתנות).

אם נניח, לדוגמה, שהשיגרה מסכמת את שני הפרמטרים שקיבלה ומדפיסה את סכומם, הרי שכאשר הפרמטרים שונים, גם התוצאה תהיה שונה.

כאשר מכריזים על שיגרה צריך לפרט את ה**פרמטרים** שתקבל, כמות וסוג:

```
Sub subname(parameter As datatype, parameter As datatype)
   statements
End Sub
```

לדוגמה,

```
Sub Sum(Num1 As Integer, Num2 As Integer)
   Print Num1 + Num2
End Sub
```

בדוגמת הכרזה זו, השיגרה Sum אמורה מקבלת שני פרמטרים ולהדפיס את סכומם. רשימת הפרמטרים כוללת את **שם** הפרמטר ואת **סוג** הפרמטר. הפרמטרים שהשיגרה מקבלת יכולים להיות משתנים מכל סוג שהוא, ואין הגבלה במספר הפרמטרים.

הקריאה לשיגרה שמקבלת פרמטרים תהיה דומה לקריאה לשיגרה ללא פרמטרים. כאשר מוגדרים פרמטרים לשיגרה, צריך לכלול אותם כאשר קוראים לשיגרה - כשמפעילים אותה, בצירוף רשימת **ארגומנטים** הנשלחים לפונקציה, על פי הפירוט בתבנית הפונקציה, ואשר השיגרה מצפה לקבל. לדוגמה:

```
Call Sum(7 ,MyNum)
```

שים לב שברשימת הארגומנטים נכלל משתנה. ניתן לשלוח כארגומנט ערכים שונים: קבוע, משתנה וגם ביטוי.

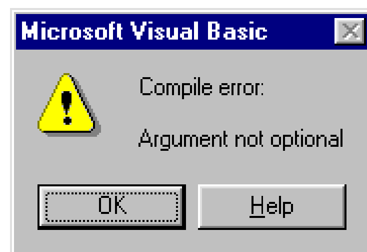
ההבדל בין **ארגומנטים לפרמטרים** הוא:

- **ארגומנטים** הם ערכים המועברים לשיגרה בזמן הקריאה לשיגרה.
- **פרמטרים** הם הערכים שהשיגרה מצפה לקבל, על פי ההכרזה שלה.

שים לב שרשימת הארגומנטים שנמסרים לשיגרה בזמן שקוראים לה נמצאת בתוך סוגריים. ניתן לוותר על סוגריים אלה, אולם במקרה זה יש **להשמיט** את ההוראה **Call**. התוצאה זהה לקריאה בדוגמה הקודמת:

```
Sum 7, MyNum
```

חשוב לציין שרשימת הארגומנטים המועברים לשיגרה חייבת להיות **זהה** לרשימת הפרמטרים שהיא מצפה לקבל (חוץ מחריגות מסוימות עליהן נעמוד בהמשך). העברת מספר ארגומנטים קטן או גדול יותר מהדרוש, תגרום לשגיאה (תרשים 10.1).



תרשים 10.1

באותה מידה, סדר הארגומנטים חייב להיות זהה לסדר הפרמטרים. לדוגמה,

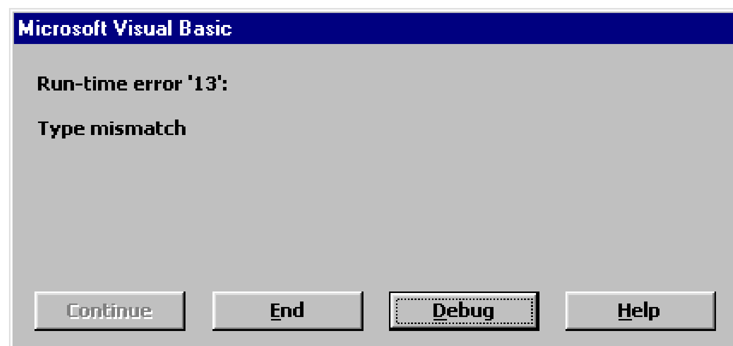
```
Call Sum(7 ,MyNum)
 ...
Sub Sum(Num1 As Integer, Num2 As Integer)
   Print Num1 +  Num2
End Sub
```

הפרמטר Num1 יקבל את הערך 7, והפרמטר Num2 יקבל את הערך המשתנה MyNum.

מלבד הסדר, חשוב לשמור גם על תאימות בסוגי המשתנים שברשימת הארגומנטים לבין רשימת הפרמטרים. אם לדוגמה, השיגרה מצפה לקבל פרמטר ראשון מסוג Integer ופרמטר שני מסוג String, אז גם רשימת הארגומנטים חייבת לכלול ארגומנט ראשון מסוג Integer וארגומנט שני מסוג String. הנה דוגמה:

```
Call Sum(7, "Danny")     'טעות
...
Sub Sum(Num1 As Integer, Num2 As Integer)
   Print Num1 + Num2
End Sub
```

הקריאה לשיגרה תגרום לשגיאה (תרשים 10.2) מכיון שאין תאימות בין רשימת הפרמטרים לבין רשימת הארגומנטים. הפרמטר השני הוא מסוג Integer, והארגומנט השני הנשלח לפונקציה הוא מסוג String.



תרשים 10.2

העברת ארגומנטים לשיגרה יכולה להיעשות בשתי דרכים: ByRef ו- ByVal.

ByVal

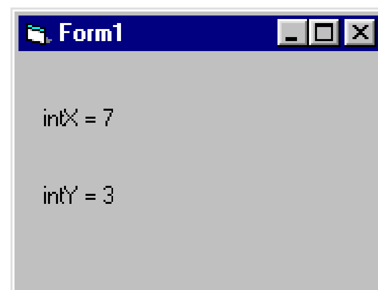
שליחת ארגומנטים **ByVal** ("**לפי ערך**") פירושה, ש**ערך** הארגומנט נשלח לשיגרה ולא הארגומנט עצמו. משמעות הדבר היא שכאשר השיגרה משנה את ערך הארגומנט בשיגרה או בפונקציה, ערך הארגומנט המקורי אינו מושפע כתוצאה מכך והוא שומר על ערכו, כי השיגרה המקבלת קולטת את הערך במשתנה פרטי שלה. לדוגמה,

```
Sub Main()
   Dim intX As Integer, intY As Integer
   intX = 7: intY = 3
   Call Swap(intX, intY)
   Print intX
   Print intY
End Sub
...
Sub Swap(ByVal Num1 As Integer, ByVal Num2 As Integer)
   Dim intHelp As Integer
   intHelp = Num1
   Num1 = Num2
   Num2 = intHelp
End Sub
```

בשיגרה Main מוצהרים המשתנים intX ו- intY ומאותחלים בערכים 7 ו- 3 בהתאמה. משתנים אלה נשלחים כארגומנטים לשיגרה Swap. הפרמטרים Num1 ו- Num2 מקבלים את ערכי הארגומנטים intX ו- intY בהתאמה. השיגרה Swap מחליפה בעזרת משתנה העזר intHelp בין ערכי המשתנים Num1 ו- Num2. בכניסה לשיגרה היו ערכי המשתנים Num1 ו- Num2 7 ו- 3 בהתאמה, ובסיום השיגרה ערך Num1 הוא 3 וערך Num2 הוא 7.

כאשר השיגרה מסיימת את ביצוע שורות הקוד שבתוכה, היא חוזרת לשורת הקוד שאחרי השורה שממנה היא נקראה. בדוגמה זו, התוכנית תחזור לשורה Print intX.

מכיון שהארגומנטים הועברו **ByVal**, רק ערכי המשתנים הועברו לשיגרה ולא המשתנים עצמם. החלפת ערכי הפרמטרים בפונקציה לא השפיעה על המשתנים intX ו-intY. משתנים אלה שמרו על ערכם. כשבשיגרה Main מתבצעות שורות ההדפסה של המשתנים intX ו- intY, נראה שהם שמרו על ערכם המקורי (תרשים 10.3).



תרשים 10.3

ByRef

שליחת ארגומנטים **ByRef** (״**לפי ייחוס**״), פירושה העברת כתובת הזיכרון של הארגומנטים אל השיגרה. במילים אחרות, אנו שולחים למעשה את המשתנים עצמם, ולכן, כל שינוי בערך הפרמטרים שייעשה בשיגרה המקבלת אותם יגרום לכך שערכם המקורי יאבד.

נשתמש בדוגמה הקודמת, אך הפעם נעביר את הארגומנטים ByRef. שים לב שהעברת ארגומנטים בדרך זו היא ברירת המחדל, ועל כן אין צורך לציין במפורש ״ByRef״:

```
Sub Main()
   Dim intX As Integer, intY As Integer
   intX = 7: intY = 3
   Call Swap(intX, intY)
   Print intX
   Print intY
End Sub
...
Sub Swap(ByRef Num1 As Integer, Num2 As Integer)
   Dim intHelp As Integer
   intHelp = Num1
   Num1 = Num2
   Num2 = intHelp
End Sub
```

במקרה זה, מכיון שהארגומנטים נשלחו ByRef, שינוי ערכי הפרמטרים בשיגרה כמוה כהחלפת ערכי הארגומנטים עצמם. כאשר התוכנית תחזור אל השיגרה Main בסיום השיגרה Swap ותדפיס את ערכי המשתנים intX ו-intY, נראה שערכי המשתנים הוחלפו.

ארגומנט רשות

רשימת הארגומנטים שהשיגרה מקבלת חייבת להיות זהה לרשימת הפרמטרים מבחינת מספרם וסוגם. אולם, ניתן לקבוע פרמטרים, שלא חייבים לשלוח אותם כשקוראים לשיגרה. לפעמים נרצה לקרוא לשיגרה עם כל הפרמטרים, ולפעמים - עם חלק מהם בלבד. במקרה זה עלינו לציין את הפרמטרים שאינם חייבים להיכלל ברשימת הארגומנטים המועברים לשיגרה כ**פרמטרי רשות**, פרמטרים אופציונליים. אין חובה להעביר לשיגרה ארגומנט, המייצג פרמטר רשות. במקרה זה רשימת הארגומנטים תהיה **קצרה** מרשימת הפרמטרים שהשיגרה מצפה לה.

כדי שפרמטר יהיה רשות יש להכריז עליו ברשימת הפרמטרים כ-Optional, כמו בדוגמה זו:

```
Sub Sum(Optional Num1 As Integer, Optional Num2 As Integer)
   Print Num1 + num2
End Sub
```

בדוגמה זו, שני הפרמטרים שהשיגרה מצפה לקבל הם רשות, אך אין חובה שכל הפרמטרים יהיו רשות. ניתן להצהיר רק על חלקם כפרמטרי רשות, כמו בדוגמה זו:

```
Sub Sum(Num1 As Integer, Optional Num2 As Integer)
   Print Num1 + num2
End Sub
```

בדוגמה זו, הפרמטר הראשון הוא חובה, והשני - רשות.

אם החלטנו בקריאה לפונקציה לא להעביר ארגומנט עבור פרמטר רשות, השיגרה תתייחס לארגומנט המייצג פרמטר רשות כמשתנה מסוג Variant שמכיל ערך Empty.

הפונקציה IsMissing

אם השיגרה מכילה ארגומנט רשות, כיצד נדע אם פרמטר זה התקבל ויש להתייחס אליו? הפונקציה **IsMissing** מסייעת לבדוק אם הארגומנט נשלח או לא נשלח. לדוגמה,

```
Sub PrintNum(Optional Num As String)
   If Not IsMissing(Num) Then
     Print Num
   End If
End Sub
```

בדוגמה זו, השיגרה ()PrintNum מקבלת פרמטר רשות. השיגרה תדפיס אותו אם ורק אם נשלח לה ארגומנט; אם לא מועבר ארגומנט, השיגרה לא תעשה דבר.

עכשיו נסה לשנות את סוג הפרמטר. רשום Integer במקום String. מה קורה עכשיו כשאתה מפעיל את הפונקציה PrintNum ולא שולח לה פרמטר?

הפתרון לכך בסעיף הבא.

ערך ברירת מחדל

הפונקציה **IsMissing** מאפשרת להתמודד עם המצב בו לא נשלח ארגומנט אופציונלי לשיגרה. במקום להשתמש בערך Empty נוכל להתעלם ממנו, אם נדע שהוא חסר. אפשרות נוספת היא להתייחס בכל מצב לארגומנט הרשות כאילו הוא נשלח, ולקבוע לו ערך ברירת מחדל אשר יהיה ערך הארגומנט אם לא הועבר לשיגרה. במצב זה, בו נקבע ערך ברירת מחדל, ערך זה יהיה ערך הארגומנט רק במידה והוא לא הועבר. אם הארגומנט הועבר, ערך זה הוא זה שהשיגרה תקבל, לדוגמה:

```
Sub PrintNum(Optional Num As Integer = 613)
   Print Num
End Sub
```

בדוגמה זו, אם יישלח ארגומנט לשיגרה ()PrintNum היא תדפיס את ערכו, במידה ולא נמסר ארגומנט, הערך שיודפס יהיה 613 (תרשים 10.4).

תרשים 10.4

רשימת הארגומנטים חייבת להיות תואמת לרשימת הפרמטרים. אם השיגרה מצפה לקבל קודם משתנה מסוג String ואחר כך משתנה מסוג Integer, זהו הסדר שחייב להיות גם ברשימת הארגומנטים. אולם, ניתן לשנות סדר זה על ידי ציון שם הפרמטר ברשימת הארגומנטים ואת ערכו. במקרה זה אין חובה לשמור על סדר סוגי המשתנים, אך חייבים לשלוח את כל הארגומנטים, אלא אם חלקם רשות. לדוגמה,

```
Sub Person(Name As String, ID As Long)
   Print Name; ID
End Sub
...
Sub Main()
   Person ID:=12345, Name:="Danny"
End Sub
```

בדוגמה זו, למרות שברשימת הפרמטרים המשתנה Name קודם למשתנה ID, בכל זאת על ידי ציון שמם וערכם, אפשר לכתוב בסדר שונה את הארגומנטים המועברים לשיגרה. שים לב שלאופרטור השוויון נוסף **נקודתיים** (=:).

מה בין שיגרה לפונקציה

כל הכללים החלים על השיגרה חלים גם על הפונקציה, החל ממבנה הפונקציה, העברת ארגומנטים וקבלת פרמטרים, ארגומנטי רשות (אופציונליים) ועוד. ההבדל התחבירי היחיד הוא, שאת מקום מילת המפתח **Sub** המגדירה שיגרה, מחליפה המילה **Function**. לדוגמה:

```
Function MyFunction()
  ...
End Function
```

על אף הדמיון הרב, יש הבדל אחד עקרוני וחשוב בין שיגרה לפונקציה והוא, היכולת של הפונקציה **להחזיר ערך**. כדי לציין את הערך שיוחזר על ידי הפונקציה, יש לציין את שם הפונקציה ואת הערך המוחזר.

מבנה הפונקציה:

```
Function functionname() As variabletype
   statements
   functionname = returnvalue
End Function
```

לדוגמה:

```
Function Sum(Num1 As Integer, Num2 As Integer) As Integer
   Dim intSum As Integer
   intSum = Num1 + Num2
   Sum = intSum
End Function
```

הערך המוחזר של הפונקציה חוזר לשיגרה או לפונקציה שקראה לה. לקבלת ערך נדרש משתנה אשר יאחסן את הערך המוחזר על ידי הפונקציה. לדוגמה:

```
Sub Main()
   Dim Result As Integer
   Result = Sum(7, 5)
   Print Result
End Sub
```

בדוגמה קוראים לפונקציה Sum עם שני ארגומנטים 7, 5. הפונקציה מקבלת ערכים אלה, מסכמת אותם ומחזירה את סכומם. הערך המוחזר מוצב (בפקודת השמה) לתוך המשתנה Result.

סוג הערך המוחזר חייב להיות זהה לסוג המוצהר בשורת הפונקציה. אם לא צוין סוג ערך המוחזר, לדוגמה:

```
Function Sum(Num1 As Integer, Num2 As Integer)
   Dim intSum As Integer
   intSum = Num1 + Num2
   Sum = intSum
End Function
```

במקרה זה, הערך המוחזר יהיה מסוג Variant. קריאה לפונקציה Sum באופן הבא **אינו** טעות:

```
Sub Main()
   Call Sum(7, 5)
End Sub
```

אין חובה לקלוט את הערך המוחזר ולהשתמש בו. הדבר היחיד שיש לשים לב אליו הוא שהמשתנה הקולט את הערך המוחזר יהיה מאותו סוג. בדוגמה זו, בה אין משתמשים בערך המוחזר, אין משמעות מיוחדת לקריאה לפונקציה, אך זו אינה טעות.

סוגי שגרות וטווח הכרה

יש שלושה סוגי שגרות ופונקציות הנבדלות בטווח ההכרה שלהם. טווח הכרה של פונקציה הוא הטווח בו הפונקציה "**מוכרת**". במילים אחרות, זהו הטווח בו ניתן לקרוא לפונקציה או לשיגרה.

שלושת סוגי השגרות / הפונקציות הם:

1. שיגרה ברמת האובייקט.
2. שיגרה ברמת הטופס.
3. שיגרה ברמת המודול.

בסעיפים הבאים נתייחס למונח "**שיגרה**", אם כי הדברים תקפים באותה מידה גם ל"**פונקציה**".

שיגרה ברמת האובייקט

כל אובייקט יודע להגיב לאירועים שונים החלים עליו. רשימת האירועים שהאובייקט יודע להגיב להם מוגדרת מראש על ידי ויז'ואל בייסיק. לחיצה על פקד מסוג לחצן היא למעשה הפעלת האירוע Click על אותו לחצן. כשהפקד TextBox מקבל מיקוד (פוקוס) וניתן לכתוב בו, מפעיל למעשה את האירוע GotFocus, וכן הלאה.

טבלה 10.1 מסכמת את האירועים הנפוצים, ואת הפעולות הגורמת להתרחשותם.

טבלה 10.1

פעולה	אירוע
לחיצה אחת על לחצן העכבר	Click
לחיצה כפולה על הלחצן	DBlClick
שחרור לחצן העכבר	MouseUp
לחיצה על לחצן העכבר ללא שחרור	MouseDown
גרירת פקד אחד על אחר	DragDrop
בתהליך גרירת פקד אחד על אחר	DragOver
הקשה על מקש	KeyPress
בזמן שחרור הלחיצה על המקש	KeyUp
בזמן שהמקש לחוץ	KeyDown
בזמן שטופס הופך לחלון פעיל	Activate
בזמן שהטופס לא פעיל (כתוצאה מהעברת הפוקוס לטופס אחר)	Deactivate
שהפקד מקבל פוקוס (מוקד)	GotFocus

LostFocus	אובדן פוקוס
Load	בזמן טעינת טופס
UnLoad	בזמן שחרור הטופס מהזיכרון
Resize	בזמן שהטופס משנה את גודלו
Change	בזמן שתוכן הפקד משתנה

כאשר יש אירוע הוא קורא לשיגרה. לדוגמה, לחיצה בעכבר (Click) על הלחצן cmdClose תגרום להצהרת השיגרה הבאה:

```
Private Sub cmdClose_Click()

End Sub
```

תפקידנו כמתכנתים לכתוב את שורות הקוד עבור הפעולות שנרצה לבצע בעת התרחשות האירוע. נניח שלחיצה על הלחצן cmdClose תסיים את התוכנית; בשיגרה Click של הפקד נרשום את מילת המפתח End המסיימת את התוכנית. לדוגמה:

```
Private sub cmdClose_click()
   End
End Sub
```

שיגרה ברמת האובייקט, אם כן, מוכרת רק לאותו אובייקט ולאירוע המסוים שחל עליו. השיגרה בדוגמה הקודמת תוכר רק כאשר נלחץ על הלחצן **cmdClose**. שים לב, כי ניתן לקרוא לשיגרה זו משיגרה אחרת, ולאו דווקא על ידי פעולת הלחיצה. במקרה זה הפעולה שתתבצע תהיה בדיוק אותה פעולה אילו היינו לוחצים על הלחצן.

לדוגמה:

```
Sub ClickEvent()
   cmdClose_Click
End Sub
```

שיגרה ברמת הטופס

שיגרה ברמת הטופס נמצאת בחלק ההצהרתי של הטופס (General Declaration). שיגרה או פונקציה ברמת הטופס מוכרים לכל האובייקטים באותו טופס. הקריאה לשיגרה או לפונקציה תהיה רגילה, היינו, ציון שם הפונקציה עם רשימת הארגומנטים והמשפט Call, אם רשימת הארגומנטים מופיעה בסוגריים.

מה בדבר אובייקטים, שגרות ופונקציות שבטפסים אחרים, האם גם הם יכולים לקרוא לפונקציה או לשיגרה ברמת הטופס? יש שתי אפשרויות להצהיר על פונקציה או שיגרה ברמת הטופס: Public ו-Private.

Public

ההצהרה Public גורמת שטווח ההכרה של שיגרה או פונקציה יהיה פרוש על כלל האובייקטים, הפונקציות והשגרות שבכל הטפסים בפרויקט. לדוגמה:

```
Public Sub PrintNum(Num As Integer)
   Print Num
End Sub
```

במקרה זה נוכל לקרוא לשיגרה לא רק מתוך הטופס שבה הוגדר, אלא גם מטופס אחר. שים לב, כי השיגרה מדפיסה את ערך המשתנה Num על גבי הטופס. כאשר תיקרא השיגרה מטופס אחר, ערך המשתנה Num יודפס על גבי הטופס בו השיגרה **מוצהרת**! מכיון שמטופס זה מתבצעת השיגרה.

הקריאה לשיגרה מטופס אחר

כאשר מוגדרת השיגרה כ-Public ניתן לקרוא לה מאובייקט, שיגרה או פונקציה שבטופס עצמו, או לחילופין מטופס אחר. כאשר הקריאה נעשית מהטופס עצמו הקריאה הרגילה היא:

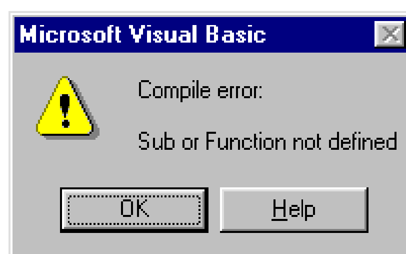
```
Private Sub cmdPrint_Click()
   Call PrintNum(7)
End Sub
```

אם הקריאה לשיגרה נעשית מטופס אחר, יש לציין לפני שם השיגרה את שם הטופס בו היא מוגדרת. לדוגמה,

```
Private Sub cmdPrint_Click()
   Call frmMain.PrintNum(7)
End Sub
```

Private

אפשרות שנייה היא להצהיר על השיגרה או הפונקציה כ-**Private**. הצהרה על שיגרה כ-Private מצמצמת את טווח ההכרה שלה לטופס שבו הוגדרה **בלבד**. ניסיון לקרוא לפונקציה Private מטופס אחר יגרום לשגיאה (תרשים 10.5). הפונקציה או השיגרה במקרה זה מוכרים אך ורק לאובייקטים, הפונקציות והשגרות בטופס בו הוגדרה בלבד.



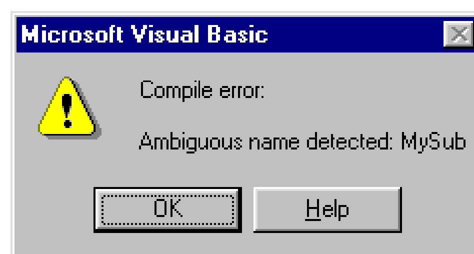
תרשים 10.5

אם הפונקציה או השיגרה הוגדרו ללא ציון מיוחד של **Private** או **Public**, ברירת המחדל של ויז'ואל בייסיק היא Public.

שיגרה ברמת המודול

טווח ההכרה של שיגרה או פונקציה ברמת המודול תלויה בסוג ההצהרה. הצהרה על השיגרה כ- **Private** תגרום לצמצום טווח ההכרה, כך שהשיגרה או הפונקציה יוכרו רק על ידי שגרות ופונקציות במודול שבו הוגדרה. לעומת זאת, הצהרה על שיגרה או פונקציה כ-**Public** תגרום להרחבת טווח ההכרה לכלל המודולים והטפסים בפרויקט וניתן יהיה לפנות אליהן מכל מקום בתוכנית. הקריאה לשיגרה או לפונקציה זהה בשני המקרים (Private או Public). אין צורך לציין לפני שם הפונקציה את שם המודול שבו הוגדרה.

אין לקרוא לשתי פונקציות או לשתי שגרות מסוג Public באותו שם, גם כשהן מוכרזות במודולים שונים. מכיון שהן מוכרות לכל הפרויקט, ויז׳ואל בייסיק לא תדע למי מהן לפנות. במקרה כזה התוכנית תופסק עם הודעת שגיאה (תרשים 10.6).



תרשים 10.6

עם זאת, כאשר יש שתי שגרות באותו שם, אחת Private והשנייה Public, קריאה לשיגרה מחוץ למודול שבו הוצהרה תגרום להפעלת זו המוגדרת כ-Public. קריאה לשיגרה מתוך המודול שבו הוצהרה, תקרא לשיגרה של אותו מודול. כלומר, יש עדיפות לשיגרה Private.

המשפט Exit

את הפקודה Exit הכרנו כאשר עסקנו בלולאות. פקודה זו גורמת ליציאה מוקדמת וישירה מלולאה. תפקיד זהה יש לפקודה זו בשיגרה או בפונקציה. הפקודה **Exit Sub** או **Exit Function** גורמת ליציאה מיידית מהשיגרה או הפונקציה. לדוגמה,

```
Sub PrintNum(Num As Integer)
   If Num > 5 Then Exit Sub
   Print Num
End Sub
```

בדוגמה זו, אם ערך הפרמטר Num גדול מ-5 התוכנית יוצאת מייד מהשיגרה, ואינה ממשיכה בה. במקרה זה, הפקודה Print Num **לא** תתבצע. הבה נעיין בפונקציה זו:

```
Function Sum(Num1 As Integer, Num2 As Integer) As Integer
   Dim intSum As Integer
   intSum = Num1 + Num2
   If intSum > 100 Then
      Sum = intSum + 10
   Else
      Sum = intSum
   End If
   MsgBox "The sum is: " & intSum
End Function
```

כאשר סכום המשתנים num1 ו- num2 גדול מ-100, הערך המוחזר של הפונקציה הוא הסכום + 10. כאשר הסכום קטן או שווה ל-100, הפונקציה מחזירה את הערך המדויק של הסכום. בכל מקרה מתבצעת השורה האחרונה של הפונקציה שבו מוצגת הודעה למשתמש על ערך הסכום. כלומר, השורה בפונקציה שבה מוחזר הערך המוחזר של הפונקציה **אינה** גורמת ליציאה מהפונקציה, והיא ממשיכה כרגיל עד **סופה**.

אם נרצה לצאת מפונקציה אחרי שהגדרנו את הערך המוחזר נשתמש בפקודה Exit. לדוגמה:

```
Function Sum(Num1 As Integer, Num2 As Integer) As Integer
   Dim intSum As Integer
   intSum = Num1 + Num2
   If intSum > 100 Then
      Sum = intSum + 10
      Exit Function
   Else
      Sum = intSum
   End If
   MsgBox "The sum is: " & intSum
End Function
```

אם סכום המשתנים num1 ו- num2 גדול מ- 100, הפונקציה תחזיר את ערך סכום המשתנים (המשתנה intSum) + 10. מייד לאחר מכן הפונקציה תסתיים. השורה האחרונה של הפונקציה **לא** תתבצע. רק במקרה בו המשתנה intSum קטן או שווה ל-100, הפונקציה תחזיר כערך את ערך המשתנה intSum ותציג את ההודעה שבשורה האחרונה בפונקציה.

תרגול

1. קלוט מספר מהמשתמש ובעזרת פונקציה קבע אם המספר הוא ראשוני או לא. אם כן הפונקציה תחזיר ערך True, ו-False במידה ולא.
2. כתוב שיגרה אשר תקלוט מחרוזת מהמשתמש ותציג אותה בסדר הפוך. לדוגמה, מהמחרוזת "שלום" תתקבל התוצאה "סולש".

11 : קבצים

כשתוכנית פועלת היא זקוקה בדרך כלל לנתונים, אלא אם היא מפצחת נוסחה מתמטית למשל, שקבועה בה מראש. הנתונים מוזנים לתוכנית על ידי המשתמש, במקלדת למשל, או שהתוכנית קוראת אותם ממאגרי הנתונים שלה שנמצאים בהתקני אחסון מגנטיים כמו למשל הדיסק הקשיח או התקליטור, והיא יכולה לקלוט אותם גם בתקשורת.

כאשר התוכנית מפיקה תוצאות או מעדכנת נתונים, היא עושה זאת בתצוגה, בדוחות מודפסים, שולחת בתקשורת ולרוב – שומרת אותם בהתקני אחסנה חיצוניים, ובעיקר, בדיסק הקשיח. למותר לציין שאי אפשר לשמור את הנתונים בזיכרון (RAM), אשר מוגבל בגודלו ותלוי באספקת חשמל לאחזקת הנתונים שבו. התקני האחסנה החיצוניים אינם תלויים באספקת חשמל וכמות הנתונים שניתן לשמור בהם גדולה מאוד.

למדנו אם כן, שמאגר הנתונים של מערכת המחשב נמצא בהתקני אחסון חיצוניים. כדי שהנתונים יהיו נגישים, עלינו לשמור אותם בסדר כלשהו, אשר מוכתב על ידי **תיקיות** (Folders), **קבצים** (Files) **ורשומות** (Records). על כך נלמד בהמשך הפרק.

סוגי קבצים

בוויז'ואל בייסיק אנו עוסקים בשלושה סוגי קבצים :

1. קובץ טקסט.
2. קובץ בינארי.
3. קובץ לגישה אקראית.

קובץ טקסט שומר את הנתונים בפורמט ASCII וניתן לקרוא את הנתונים השמורים בו בקריאת טקסט רגיל. לעומתו, **קובץ בינארי** שומר את הנתונים בצורת ערכים בינאריים.

לדוגמה, המספר 1234 יכול להישמר בקובץ בשני אופנים. אפשרות אחת היא לשמור אותו בקובץ טקסט, שבו כל תו נשמר כמות שהוא וכמות הבתים שיידרשו לשמירה של המספר 1234 היא 4 בתים, בית לכל סיפרה.

בקובץ בינארי דרושים 2 בתים בלבד לאחסנת המספר, מכיון שההוא מסוג Integer שתופס 2 בתים בלבד. הייצוג של המספר בזיכרון יהיה בינארי (בהדפסה אנו רואים זאת בייצוג הקסדצימלי). כשנערוך את תוכן הקובץ ונקרא אותו, לא נראה בו את 4

הספרות 1234 (כבקובץ טקסט), אלא את הייצוג ההקסדצימלי שלהן. כדי לקרוא קובץ בינארי יש לדעת בדיוק באיזה מבנה הוא נשמר, מכיון שתוכנו אינו קריא כמו קובץ טקסט רגיל.

בפרק זה נעסוק בקבצי **טקסט**. על קבצים בינאריים וקבצים לגישה אקראית לא נעסוק בספר, מכיון שבדרך כלל שמירה שאינה בקובץ טקסט נעשית במסד נתונים, המהווה תחלופה יעילה לשמירת נתונים, ואשר תדון בפירוט בפרק 16.

קבצי טקסט

קבצי טקסט הם כל סוגי הקבצים אשר שומרים את הנתונים בפורמט ASCII, כדוגמת הקבצים בעלי הסיומת txt, ini, dat, bat וכו׳. קבצים מסוג זה ניתן לערוך בעזרת כל תוכנית עורך (Editor) של קבצי טקסט. ניתן גם לערוך באופן דינמי קבצי טקסט מתוך שורות קוד בוויז׳ואל בייסיק, ועל כך בהמשך הפרק.

פתיחת קובץ

כדי שנוכל לערוך קובץ טקסט ולבצע בו שינויים, לקרוא ממנו או להוסיף לו נתונים, יש ל״פתוח״ אותו תחילה ולמעשה – לעשותו זמין לתוכנית. פתיחת קובץ נעשית בעזרת מילת המפתח **Open**.

תחביר ההוראה לפתיחת קובץ:

```
Open path\filename For aecesstype As filenumber
```

לדוגמה:

```
Open "C:\Autoexec.bat" For Input As #1
```

בדוגמה זו נפתח הקובץ Autoexec.bat לצורך קריאה כקובץ שסימנו הוא **1** (#1). אם נפתח קובץ נוסף, ניתן לו מספר שונה (אלא אם סגרנו את הקובץ הראשון, ואז ניתן להשתמש שוב באותו מספר).

מספר הקובץ הוא אמצעי זיהוי לקובץ. כאשר פותחים כמה קבצים במקביל ורוצים לבצע בהם פעולות כלשהן, עלינו להשתמש במספר הקובץ כדי לשייך את הפעולה לקובץ המתאים.

סוגי פתיחת קובץ טקסט

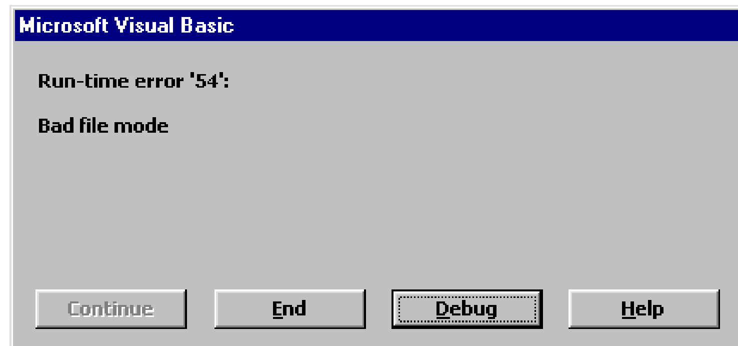
קובץ טקסט אפשר לפתוח בשלושה אופנים:

1. **Input** - פותחים את הקובץ ל**קריאה בלבד**.

לדוגמה:

```
Open "C:\Autoexec.bat" For Input As #1
```

בדוגמה זו פותחים את הקובץ Autoexec.bat לקריאה בלבד. כלומר, ניתן לקרוא ממנו נתונים, אך לא לכתוב בו או לעדכן. ניסיון לכתוב בקובץ שהוגדר למצב קריאה בלבד יגרור הודעת שגיאה (תרשים 11.1).

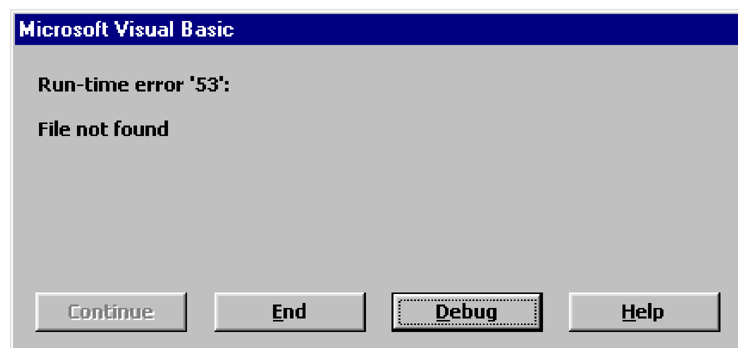


תרשים 11.1

את שם הקובץ ואת הנתיב יכול להחליף משתנה מחרוזת המכיל את השם ואת הנתיב של הקובץ שברצוננו לפתוח. לדוגמה,

```
Sub Main()
   Dim strFileName As String
   strFileName = "c:\Autoexec.bat"
   Open strFileName For Input As #1
End sub
```

אם קובץ זה אינו קיים תופיע הודעת שגיאה (תרשים 11.2).



תרשים 11.2

2. **Output** - פתיחת הקובץ **לכתיבה**.

לדוגמה:

```
Open "c:\Mytext.txt" For Output As #2
```

אם קובץ זה קיים, פתיחה לצורך כתיבה **מוחקת** את הנתונים הקודמים שהיו בו. אם לא קיים קובץ בשם זה, ויז'ואל בייסיק תיצור אותו ותאפשר כתיבה בו.

3. **Append** - פתיחת הקובץ **להוספה**. קובץ מסוג זה מאפשר קריאה וכתיבה.

לדוגמה:

```
Open "c:\Autoexec.bat" For Append As #3
```

פתיחת קובץ במצב זה מאפשרת להוסיף נתונים חדשים לנתונים הקיימים. בניגוד למצב פתיחה לכתיבה, הנתונים הקודמים אינם נמחקים.

קריאה מקובץ

לאחר שפותחים קובץ במצב קריאה ניתן לקרוא ממנו נתונים בשתי דרכים:

1. קריאת שורה שלמה.
2. קריאת מספר תווים מוגדר.

כאשר פותחים קובץ, מצביע הקלט מוצב בראש הקובץ.

קריאת "שורה שלמה" פירושה קריאת כל התווים עד תו "סוף שורה". קריאה של 10 תווים תקרא את עשרת התווים הראשונים של הקובץ.

קריאת שורה שלמה נעשית באמצעות המשפט **Line Input**. משפט זה קורא את השורה **עד** לתו "carriage return" שנוצר בפעולת מקש Enter, ואינה כוללת אותו. תו זה מסמן את סוף השורה **ואין** הוא נכלל במחרוזת שקורא המשפט Line Input. משפט זה מחזיר כערך את תוכן השורה הנוכחית בקובץ, ומקדם את מצביע הקלט לשורה הבאה. תחביר המשפט הוא:

```
Line Input filenumber, variable
```

המשתנה **String** מכיל את הערך המוחזר של המשפט. לדוגמה:

```
Dim strResult As String
Open "c:\Autoexec.bat" For Input As #1
Line Input #1, strResult
Print strResult
Close #1
```

בדוגמה זו המשתנה strResult מכיל את תוכן השורה הראשונה של הקובץ Autoexec.bat, ומצביע הקובץ עומד עתה בראש השורה השנייה. כאשר נדפיס את תוכן המשתנה strResult, תוצג בטופס השורה הראשונה של הקובץ (תרשים 11.3).



תרשים 11.3

הפונקציה input

אפשרות נוספת לקרוא מקובץ היא קריאה של מספר תווים מוגדר. באפשרות זו ניתן להגדיר מספר תווים רצוי מתוך שורה בקובץ, או לקרוא יותר מאורך שורה אחת. קריאה באופן זה נעשית בעזרת הפונקציה **Input**. תחביר הפונקציה:

```
variable = Input(number of characters, filenumber)
```

המשתנה המקבל את הערך המוחזר של הפונקציה הוא מסוג **String**. לדוגמה:

```
Dim strResult As String
Open "c:\Autoexec.bat" For Input As #2
strResult = Input(30, #2)
Print strResult
Close #2
```

בדוגמה זו יודפסו 30 התווים הראשונים של הקובץ autoexec.bat ומצביע הקלט יעמוד על התו ה-31. פקודת קריאה נוספת של 30 תווים מהקובץ תגרום לקריאת 30 התווים **הבאים** וכן הלאה.

בפונקציה זו אפשר להשתמש כאשר מבנה הקובץ קבוע וידוע מראש. קריאת 30 תווים תהיה משמעותית אם נדע מה הם מכילים, אך אם מבנה הקובץ אינו ידוע, לא תהיה לכך כל משמעות, כי אולי 30 התווים הם חלק משורה או פיסקה, ואולי אף נחתכים באמצע מילה. במקרה זה עדיף לקרוא שורות שלמות באמצעות המשפט Line Input.

הפונקציה seek

כאשר מבנה הקובץ ידוע, אפשר לקרוא מספר תווים רצוי כדי לקבל את הנתונים הדרושים מתוך הקובץ. הקריאה נעשית בעזרת הפונקציה **Input** באופן סדרתי; כלומר, כשהקובץ נפתח מצביע הקלט מוצב בראשו. פקודת קריאה של 20 תווים, למשל, תגרום לקריאת 20 תווים החל מהמקום שבו נמצא המצביע,ואת 20 התווים הבאים נקרא בהמשך, באופן סדרתי.

כשנרצה לקרוא 20 תווים במקום מוגדר אחר, למשל החל בתו ה-50, נשתמש בפונקציה **Seek** כדי להציב את מצביע הקלט במקום מוגדר אחר בקובץ ולקרוא ממנו באמצעות הפונקציה Input. כארגומנטים נמסור לפונקציה Seek את מספר הקובץ ואת הערך המציין את המקום שממנו רוצים לקרוא.

תחביר הפונקציה:

```
Seek filenumber, position
```

לדוגמה:

```
Seek #1, 50
```

השיגרה הבאה מדפיסה את תוכן השורה הראשונה של הקובץ `Autoexec.bat` (ראה תרשים 11.4):

```
Private Sub Form_Click()
    Dim strResult As String
    Open "c:\Autoexec.bat" For Input As #1
    Line Input #1, strResult
    Print strResult
    Close #1
End Sub
```



תרשים 11.4

בשיגרה הבאה, נשתמש במשפט Seek כדי להציב את המצביע על התו השני בקובץ וממקום זה נקרא ונדפיס שלושה תווים. בעזרת הפונקציה Input נקרא את שלושת התווים הבאים (ראה תרשים 11.5).

```
Private Sub Form_Click()
    Dim strResult As String
    Open "c:\autoexec.bat" For Input As #1
    Seek #1, 2
    Print Input(3, #1)
    Close #1
End Sub
```



תרשים 11.5

כתיבה לקובץ

פתיחת קובץ לכתיבה נעשית באמצעות הפקודה **Output** ולפתיחת קובץ להוספה נשתמש בפקודה **Append**.

כתיבה לקובץ נעשית באמצעות הפקודה **Print**. תחביר הפונקציה:

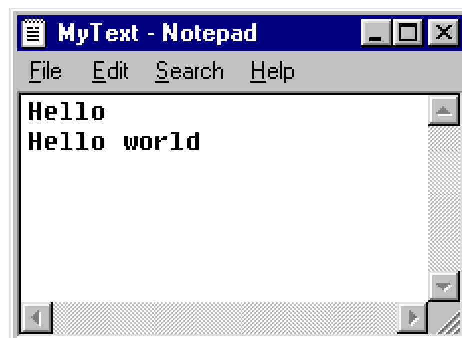
```
Print filenumber, stringlist
```

את המחרוזת הנשלחת לכתיבה ניתן להחליף במשתנה מסוג **String**. לדוגמה,

```
open "c:\MyText.txt" For Output As #1
Print #1, "Hello"
Print #1, "Hello"; " "; "world"
close #1
```

לאחר כתיבת המחרוזת לקובץ, המשפט **Print** גורם באופן אוטומטי למעבר לשורה חדשה. במילים אחרות, כל מחרוזת נכתבת בשורה נפרדת, אלא אם המשפט Print כולל מספר מחרוזות. במקרה זה, כל המחרוזות תיכתבנה בשורה אחת בקובץ.

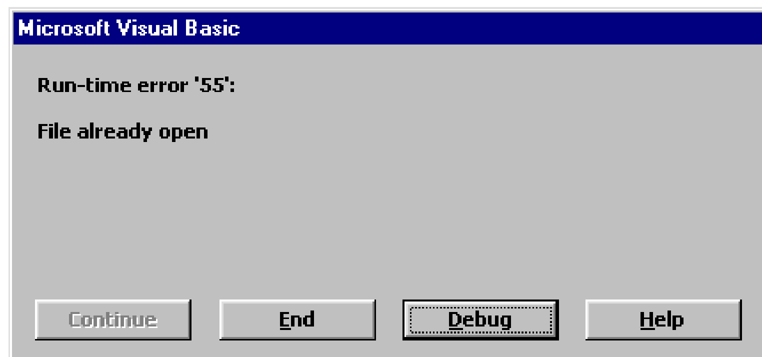
תרשים 11.6 מציג את תוכן הקובץ שנוצר מביצוע שורות הקוד שבדוגמה לעיל. שים לב לתו הרווח (" ") שבין שתי המחרוזות, כי ויזואל בייסיק אינה יודעת להציב אותו בעצמה.



תרשים 11.6

סגירת קובץ

לפני סיום התוכנית צריך לסגור כל קובץ שנפתח במהלך התוכנית. אם רוצים לפתוח קובץ כלשהו לפעולות שונות, כמו למשל לקריאה ואחר כך להוסיף לו נתונים, צריך לסגור אותו בסיום פעולות הקריאה, לפני שפותחים אותו להוספה. אם פותחים קובץ ולא סוגרים אותו, לא ניתן לפתוח אותו שוב (תרשים 11.7).



תרשים 11.7

כדי לפתוח את הקובץ שנית במצב חדש, כלומר לביצוע פעולות אחרות, יש לסגור אותו ורק אחר כך לפתוח אותו שנית.

תחביר הפקודה לסגירת קובץ:

```
Close filenumer
```

לדוגמה:

```
Private Sub Form_Click()
  Open "c:\MyText.txt" For Output As #1
  Print #1, "Hello"
  Print #1, "Hello"; " "; "world"
  Close #1
End Sub
```

לאחר שהקובץ נסגר ניתן להשתמש שוב במספר הקובץ שלו גם עבור קובץ אחר. הכלל הוא, שאפשר לתת לקובץ רק מספר זיהוי שאינו בשימוש.

פונקציות לעבודה עם קבצים

מלבד פתיחת קובץ למטרות שונות וסגירתו, יש בוויז'ואל בייסיק מספר פונקציות לעבודה עם קבצים. פונקציות אלו שימושיות לכל סוגי הקבצים, ואינן מוגבלות לקבצי טקסט או לקבצים אחרים.

הפונקציה DIR

הפונקציה **DIR** מאפשרת לחפש קובץ על פי שמו ומאפייניו. הפונקציה מקבלת כפרמטרים את שם הקובץ ואת מאפייניו, ומחזירה את שם הקובץ **הראשון** שמתאים לתנאי החיפוש.

תחביר הפונקציה:

```
Dir(search filename, attributes)
```

לדוגמה:

```
Result = Dir("c:\autoexec.bat", vbReadOnly)
```

לפניך חלק מהערכים שמקבל הארגומנט **Attribute**:

vbReadOnly - קובץ קריאה בלבד.

vbHidden - קובץ מוסתר.

vbArchive - קובץ רגיל.

vbSystem - קובץ מערכת.

VbDirectory - תיקיה.

ניתן לשלב בחיפוש כמה מאפיינים שדרוש, אך צריך ל"חבר" ביניהם בעזרת התו +.

לדוגמה:

```
Print Dir("C:\Windows\System.ini", vbArchive + vbReadOnly)
```

במקום שם קובץ ניתן לציין משפחה של קבצים המאופיינים על ידי הסיומת, למשל bat:

```
Result = Dir("c:\*.bat", vbArchive)
```

בדוגמה זו שם הקובץ **הראשון** מסוג bat יוחזר כערך על ידי הפונקציה, ללא תלות בשמו.

והנה דוגמה נוספת להדפסת שם הקובץ הראשון שיימצא, אשר שמו מתחיל ב- A והסיומת שלו - bat:

```
Result = Dir("c:\a*.bat", vbArchive)
Print Result
```

הפונקציה FileLen

הפונקציה **FileLen** מקבלת כפרמטר שם קובץ (ונתיבו), ומחזירה את גודלו בבתים.
תחביר הפונקציה הוא:

```
FileLen(filename)
```

לדוגמה:

```
Print FileLen("c:\windows\system.ini")
```

הפונקציה LOF

הפונקציה **LOF** מחזירה את הגודל בבתים של קובץ פתוח. כפרמטר היא מקבלת את מספר הקובץ שניתן בעת שפתחו אותו.

תחביר הפונקציה:

```
LOF(filenumber)
```

לדוגמה:

```
Open "c:\autoexec.bat" For Input As #1
MsgBox LOF(1)
```

הפונקציה EOF

הפונקציה **EOF** מזהה את סוף הקובץ. הפונקציה מחזירה ערך **True** כאשר מצביע הקריאה או הכתיבה מגיע לסוף הקובץ. כפרמטר היא מקבלת את מספר הקובץ.

תחביר הפונקציה:

```
EOF(filenumber)
```

שימוש נפוץ בפונקציה EOF הוא בעת קריאה מקובץ שגודלו אינו ידוע. קוראים את הקובץ שורה אחר שורה עד סופו, וכשמגיעים לסוף הקובץ מפסיקים לקרוא ממנו. המידע שמגיעים לסוף הקובץ מתקבל על ידי הפונקציה EOF. לדוגמה,

```
Open "c:\autoexec.bat" For Input As #1
Dim Result As String
Do While Not EOF(1)
   Line Input #1, Result
   Print Result
Loop
Close #1
```

בדוגמה זו מודפס תוכן הקובץ Autoexec.bat בטופס (תרשים 11.8).

```
Form1
PROMPT $p$g
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\DOS;c:\nc
rem SET SOUND=C:\SB16
rem SET BLASTER=A220 I5 D1 H5 P330 T6
rem SET MIDI=SYNTH:1 MAP:E
rem SET CTCM=C:\CTCM
rem SET TEMP=C:\DOS
rem LH C:\SB16\DIAGNOSE /S
rem LH C:\SB16\MIXERSET /P /Q
rem LH C:\CTCM\CTCU /S
rem LH mouse
LH C:\WINDOWS\COMMAND\MSCDEX.EXE /D:BTCCD001 /V
mode con codepage prepare=((862) C:\WINDOWS\COMMAND\hebega.cpi)
mode con codepage select=862
```

תרשים 11.8

בתוכנית קוראים את הקובץ שורה אחר שורה בלולאה, אשר גם מדפיסה את השורות האלו. הלולאה ממשיכה לקרוא מהקובץ כל עוד לא מגיעים לסוף הקובץ, או במילים אחרות כל עוד הפונקציה EOF מחזירה ערך **False**.

הפונקציה FileCopy

באמצעות הפונקציה **FileCopy** ניתן להעתיק קובץ אל קובץ אחר. הפונקציה מקבלת כפרמטרים שתי מחרוזות המייצגות שם ונתיב קובץ המקור ושל קובץ היעד.

תחביר הפונקציה:

```
FileCopy(source, destination)
```

לדוגמה:

```
FileCopy("c:\MyText.txt", "c:\Autoexec.bat")
```

ניתן לשלוח כארגומנטים שני משתנים מסוג String, המכילים את שם הקובץ ואת הנתיב שלו. לדוגמה,

```
Dim strSource As String
Dim strDest As String
strSource = "c:\MyText.txt″
strDest = "c:\autoexec.txt″
FileCopy strSource, strDest
```

הפונקציה FileDateTime

הפונקציה **FileDateTime** מקבלת כפרמטר שם קובץ ונתיב ומחזירה **variant** המכיל את התאריך והשעה בו הקובץ נוצר או עודכן לאחרונה.

תחביר הפונקציה:

```
FileDateTime(pathname)
```

לדוגמה:

```
Dim Result
Result = FileDateTime("c:\windows\system.ini")
```

הפונקציה Loc

הפונקציה **Loc** מקבלת כפרמטר את מספר הקובץ הפתוח, ומחזירה כערך משתנה מסוג **Long**, המייצג את מיקומו של מצביע הקלט (או מצביע הפלט) בקובץ.

תחביר הפונקציה:

```
Loc(filenumber)
```

לדוגמה:

```
Open "c:\MyText.txt" For Input As #1
Print Loc(1)
Close #1
```

הפונקציה FreeFile

הפונקציה **FreeFile** מחזירה ערך מסוג Integer הקובע מהו מספר הקובץ הבא הפנוי לשימוש עבור קובץ חדש. כל קובץ שנפתח מקבל מספר ייחודי, וכל עוד הוא לא נסגר, אי אפשר לתת לקובץ נוסף שנפתח את אותו מספר. כדי לדעת איזה מספר נוכל לתת לקובץ שנרצה לפתוח, נשתמש בפונקציה FreeFile המחזירה כערך מספר אפשרי לקובץ. הפונקציה מחזירה ערך ואין לה ארגומנטים.

לדוגמה:

```
Dim intFileNumber As Integer, I As Integer
For I = 1 To 10
    intFileNumber = FreeFile
    Open "c:\MyText.txt" For Append As #intFileNumber
    Print #intFileNumber, "Line:" & I
    Close #intFileNumber
Next I
```

הפונקציה GetAtt

הפונקציה **GetAtt** מקבלת כערך שם קובץ ואת נתיבו ומחזירה את מאפייני הקובץ כגון: מוסתר, ארכיון וכדו׳.

תחביר הפונקציה:

```
GetAtt(Pathname)
```

לדוגמה:

```
MsgBox GetAtt("c:\Autoexec.bat")
```

הערכים המוחזרים על ידי הפונקציה GetAtt מייצגים את הקבועים המוצגים בטבלה 11.1:

טבלה 11.1

Constant	Value	Description
vbNormal	0	Normal
vbReadOnly	1	Read-only
vbHidden	2	Hidden
vbSystem	4	System
vbDirectory	16	Directory or folder
vbArchive	32	File has changed since last backup

הפונקציה SetAtt

הפונקציה **SetAtt** קובעת את תכונות הקובץ. אם נרצה לקבוע את מאפיין הקובץ לקריאה בלבד, נעשה זאת בעזרת פונקציה זו. הפונקציה מקבלת שני פרמטרים: את שם הקובץ ונתיבו, ואת המאפיין.

תחביר הפונקציה הוא:

```
SetAtt(Pathname.attribute)
```

לדוגמה:

```
SetAtt("MyText.txt", vbReadOnly)
```

רשימת הקבועים שהפונקציה מקבלת בפרמטרים מוצגים בטבלה 11.2:

טבלה 11.2

Constant	Value	Description
vbNormal	0	Normal (default)
vbReadOnly	1	Read-only
vbHidden	2	Hidden
vbSystem	4	System file
vbArchive	32	File has changed since last backup

פונקציות לעבודה עם מחרוזות

הפונקציות לעבודה עם מחרוזות שימושיות לפעולות תכנות שונות, ולאו דווקא לעבודה בקבצים. הסיבה שהן מופיעות דווקא בפרק העוסק בקבצים היא, שבדרך כלל פועלים על שם הקובץ שהוא מחרוזת, או משתנה מחרוזת. פעולות אלו כוללות שרשור הנתיב ושם הקובץ לשם אחד וכו'. בסעיפים הבאים נלמד לבצע פעולות שונות על מחרוזות ולקבל מידע מוגדר מתוך מחרוזות אלו.

הפונקציה Len

הפונקציה **Len** מקבלת כפרמטר מחרוזת, או משתנה String המכיל מחרוזת, ומחזירה את אורכה בתווים.

תחביר הפונקציה הוא:

```
Len(String)
```

לדוגמה:

```
Dim strName AS String
strName = "Danny"
Print Len(strName)
```

בדוגמה זו יודפס הערך 5 בטופס; זהו אורך המחרוזת שנמצא במשתנה strName.

הפונקציה Right

הפונקציה **Right** מחזירה מספר תווים בצד הימני של המחרוזת. הפונקציה מקבלת כפרמטרים את המחרוזת ואת מספר התווים הרצוי ומחזירה מחרוזת המכילה את מספר התווים המבוקשים בצד ימין.

תחביר הפונקציה:

```
Right(String, length)
```

לדוגמה:

```
Dim strName As String
strName = "Danny"
Print Right(strName, 2)
```

על הטופס יוצגו 2 התווים הימנים של המחרוזת שמכיל המשתנה strName. המחרוזת שתוצג היינה "ny".

הפונקציה Left

הפונקציה **Left** מחזירה מספר תווים רצויים מתוך מחרוזת, אך הפעם התווים המוחזרים נמצאים בצד שמאל של המחרוזת.

תחביר הפונקציה הוא:

```
Left(String, length)
```

לדוגמה:

```
Dim strName AS String, intLen As Integer
strName = "Danny"
intLen = 2
Print Left(strName, intLen)
```

במקרה זה יוצגו על גבי הטופס התווים "Da".

הפונקציה Mid

הפונקציה **Mid** מחזירה קבוצה חלקית של תווים מתוך מחרוזת שלמה. הפרמטרים שמקבלת הפונקציה הם: המחרוזת השלמה, המיקום ההתחלתי במחרוזת שממנו נתחיל לקרוא את התווים ומספר התווים הרצוי.

תחביר הפונקציה הוא:

```
Mid(string, start, length)
```

לדוגמה:

```
Dim strName As String
Dim intStart As Integer, intLen As Integer
strName = "Danny"
intStart = 2 : intLen = 3
Print Mid(strName, intStart, intLen)
```

שורות קוד אלו יציגו על הטופס את התווים "ann".

שים לב שכאשר ערך הפרמטר Start גדול מאורך המחרוזת כולה, הפונקציה תחזיר מחרוזת ריקה (" ").

המשפט Mid

המשפט **Mid** מחליף קטע מחרוזת בקטע אחר.

תחביר המשפט הוא:

```
Mid(stringtomodify, start, length) = string
```

הפרמטרים שמקבל המשפט הם: המחרוזת שתשתנה, מיקום התחלתי של השינוי ואורך התווים שיוחלפו. המשתנה stringtomodify מייצג את המחרוזת אשר תחליף את קטע המחרוזת הקיים. לדוגמה:

```
Dim MyString
MyString = "The dog jumps"
Mid(MyString, 5, 3) = "fox"
```

לאחר השינוי, תוכן משתנה המחרוזת יהיה: "MyString = "The fox jumps.

הפונקציה InStr

הפונקציה **InStr** מחפשת תו או תת-מחרוזת בתוך מחרוזת. אם התו או תת-המחרוזת נמצאים בתוך המחרוזת, הפונקציה מחזירה משתנה **Long** שערכו מייצג את מיקום התו או תת-המחרוזת במחרוזת השלמה. אם התו או תת-המחרוזת לא נמצאים, הפונקציה מחזירה את הערך 0.

תחביר הפונקציה הוא:

```
InStr (start, SearchString, SearchChar, Compare)
```

הפרמטרים שהפונקציה יכולה לקבל:

Start - המיקום במחרוזת (הראשית) ממנו יתחיל החיפוש.

SearchString - המחרוזת הראשית.

SearchChar - התו או תת-המחרוזת.

Compare - סוג השוואה.

לדוגמה:

```
Dim intStart AS Integer
Dim strSearchString AS String
Dim strSearchChar As String
intStart = 1
strSearchString = "Israel"
strSearchChar = "a"
MsgBox InStr(intStart, strSearchString, strSearchChar, 1)
```

בדוגמה זו, מיקום התו "a" בתוך המחרוזת "Israel" הוא במקום הרביעי, ולכן תיבת ההודעה תציג את הערך 4.

הערכים המוחזרים על ידי הפונקציה מוצגים בטבלה 11.3:

טבלה 11.3

If	InStr returns
string1 is zero-length	0
string1 is **Null**	Null
string2 is zero-length	*start*
string2 is **Null**	Null
string2 is not found	0
string2 is found within	Position at which match is found *string1*
start > *string2*	0

לפניך דוגמאות לסוגי השוואה שונים:

```
Dim SearchString, SearchChar, MyPos

SearchString ="XXpXXpXXPXXP"  'String to search in.
SearchChar = "P"  'Search for "P"

'A textual comparison starting at position 4. Returns 6.
MyPos = InStr(4, SearchString, SearchChar, 1)

'A binary comparison starting at position 1. Returns 9.
MyPos = InStr(1, SearchString, SearchChar, 0)

'Comparison is binary by default (last argument is omitted).
MyPos = InStr(SearchString, SearchChar)   'Returns 9.

MyPos = InStr(1, SearchString, "W") 'Returns 0.
```

הפונקציה Trim

הפונקציה **Trim** מקבלת כפרמטר מחרוזת, ואם יש רווחים, היא מסירה אותם מימין ומשמאל למחרוזת.

שים לב שהפונקציה אינה מורידה רווחים **בתוך** המחרוזת, כאשר היא מכילה למשל שתי מילים ורווח ביניהם. פונקציה זו מסירה רווחים מיותרים מימין ומשמאל למחרוזת בלבד.

תחביר הפונקציה הוא:

```
Trim(String)
```

לדוגמה:

```
MsgBox Trim("   Jerusalem   ")
```

במקרה זה תציג תיבת ההודעה את המחרוזת Jerusalem בלבד, ללא הרווחים שמימין ומשמאל.

הפונקציה RTrim

הפונקציה **RTrim** מנקה רווחים מיותרים מימין למחרוזת. הפונקציה מקבלת כפרמטר מחרוזת ומחזירה אותה ללא הרווחים מימינה. אם יש רווחים מיותרים משמאל למחרוזת, הפונקציה אינה מסירה אותם. הפונקציה מתייחסת אך ורק לרווחים שמימין למחרוזת.

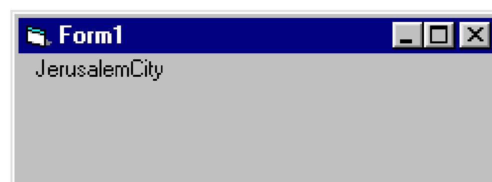
תחביר הפונקציה הוא:

```
RTrim(String)
```

לדוגמה:

```
Print RTrim("   Jerusalem   ") & "City"
```

הפלט שיודפס על גבי הטופס יכלול את המחרוזת Jerusalem ללא רווחים מימין וצמוד לה – המחרוזת City (תרשים 11.9).



תרשים 11.9

הפונקציה LTrim

הפונקציה LTrim מסירה רווחים מיותרים שנמצאים משמאל למחרוזת.

תחביר הפונקציה הוא:

```
LTrim(String)
```

לדוגמה:

```
Dim strFirstName AS String
Dim strLastName As String
strFirstName = "   Danny   "
strLastName = "Cohen"
MsgBox LTrim(strFirstName) & strLastName
```

תיבת ההודעה תציג את שתי המחרוזות כשיש רווח ביניהם, הנובע מכך שהרווחים שמימין למחרוזת Danny לא הוסרו (תרשים 11.10). הרווח משמאל למחרוזת הראשונה נמחק.



תרשים 11.10

הפונקציה UCase

הפונקציה **UCase** מסבה מחרוזת הכתובה אנגלית באותיות קטנות לאותיות גדולות.

תחביר הפונקציה הוא:

```
UCase(String)
```

לדוגמה, כדי להציג את המחרוזת "Danny" באותיות גדולות (תרשים 11.11) נכתוב כך:

```
MsgBox UCase("Danny")
```



תרשים 11.11

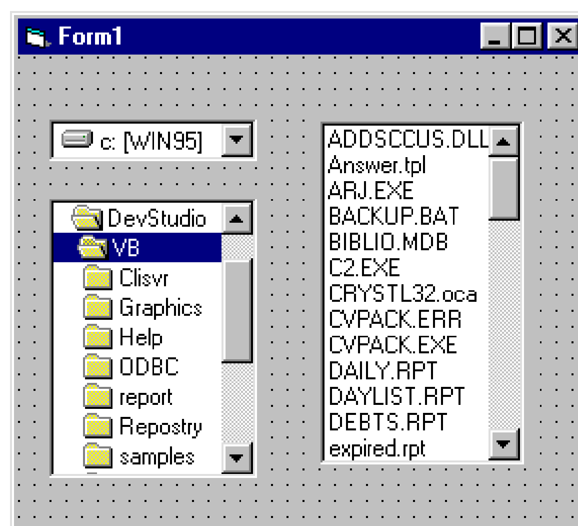
שים לב לדוגמה הבאה:

```
MsgBox UCase("DANNY")
```

במקרה זה, המחרוזת אינה משתנה. הפונקציה UCase גורמת לאותיות להפוך מקטנות לגדולות, ולא להיפך.

פקדים למיפוי דיסק

בוויז'ואל בייסיק קיימים שלושה פקדים שמאפשרים לקבל מידע על הכוננים, התיקיות והקבצים. השימוש בפקדים אלה הוא משולב (תרשים 11.12), כי שלושתם יחד נותנים תמונה שלמה על מערך הזיכרון החיצוני של המחשב. עם זאת, ניתן גם להשתמש בכל פקד בנפרד ללא קשר לאחרים.



תרשים 11.12

שים לב שיש אפשרות נוספת להציג בפני המשתמש את תוכן הדיסק ואת היכולת לדפדף בתיקיות. אפשרות זאת ניתנת על ידי תיבות דו-שיח מיוחדות עליהן נדון בהמשך הפרק.

בעזרת הפקדים למיפוי דיסק נוכל לבנות תיבות דו-שיח מותאמות, המאפשרות עיון בתכולת האחסנה החיצונית. המידע המופיע בפקדים אלה מתקבל ממערכת ההפעלה ומציג באופן אוטומטי את כל המידע, אך גם ניתן להגדיר דרך מאפייני הפקדים מהו המידע המסוים שיוצג בהם. גמישות זו להצגת מידע אינה מותרת בתיבות הדו-שיח הרגילות, ובזה יתרונם של פקדים אלה על תיבות הדו-שיח.

הפקד DriveListBox

הפקד **DriveListBox** מציג את רשימת הכוננים הקיימים במערכת המשתמש. הכוננים נמצאים בתוך רשימה נפתחת אשר מתוכה יכול המשתמש לבחור בכונן הרצוי. לחיצה על אחד הכוננים ברשימה גורמת לבחירתו ולהצגתו בתיבת הטקסט שבראש התיבה הנפתחת.

את בחירת המשתמש ניתן לקרוא בעזרת המאפיין Drive של הפקד. לדוגמה,

```
MsgBox Drive1.Drive
```

את שם הכונן שיוצג ניתן לקבוע גם בדרך של תכנות. דוגמה:

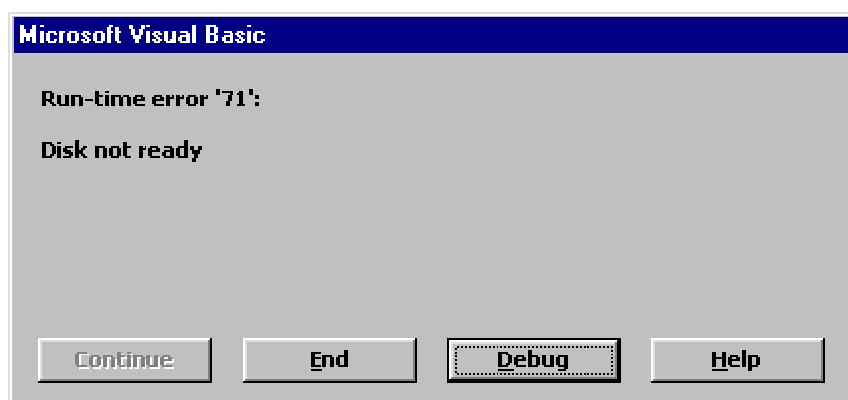
```
Drive1.Drive = "c:\"
```

שים לב שבחירת אחד הכוננים מהרשימה רק מדגישה אותו, אבל אינה משנה בפועל את הכונן הפעיל הנוכחי, למרות שהמאפיין **Drive** מקבל את הבחירה כערך. שינוי בפועל של כונן נעשה בעזרת המשפט **ChDrive**. דוגמה:

```
ChDrive Drive1.Drive
```

רק אחרי שורת קוד זו, משתנה כונן העבודה הפעיל של המערכת על פי הבחירה.

בבחירת הכונן יכול להיווצר מצב שגיאה, שבו המשתמש בוחר בכונן :A, למשל, ואין דיסקט בתוכו. במצב זה התוכנית תופסק עם הודעת שגיאה (תרשים 11.13). זה המקום לבצע טיפול בשגיאות ולוודא ששגיאה כזו תטופל בהתאם ותמנע הפסקה לא מתוכננת של התוכנית (על הטיפול בשגיאות נדון בפרק 20).



תרשים 11.13

הפקד DirListBox

הפקד **DirListBox** מציג את מבנה התיקיות של הכונן הנוכחי, ומדגיש את התיקיה הפעילה. לפקד DirListBox יש מאפיין הקובע אילו תיקיות ובאיזה כונן יוצגו בו. מאפיין זה הוא **Path** (נתיב). כדי לקבוע שהתיקיות שיקבל Path תהיינה של הכונן שנבחר באמצעות הפקד DriveListBox, נכתוב את שורת הקוד הבאה:

```
Dir1.Path = Drive1.Drive
```

שורת קוד זו קובעת מה יוצג בפקד DirListBox, אך היא אינה משנה את התיקיה הנוכחית, הפעילה. כדי לשנות בפועל את תיקיית העבודה נשתמש במשפט **ChDir**. לדוגמה,

```
ChDir Dir1.Path
```

נוכל לקבוע שכאשר תוצג רשימת תיקיות בעזרת הפקד DirListBox, תיקיית ברירת המחדל תהיה התיקיה של היישום שלנו, לצורך זה נשתמש במשפט ChDir יחד עם נתיב תיקיית היישום. אך מהי תיקיית היישום? גם אם נקבע לה נתיב מוגדר מראש, כמו למשל "C:\My Program" ונגדיר אותה כברירת מחדל, המשתמש יכול לשנות אותה לתיקיה אחרת כרצונו. על כן, נשתמש באובייקט **App** המייצג את היישום שלנו, ואשר יש לו מאפיינים וביניהם המאפיין Path. המאפיין **Path** קובע את התיקיה שבה נמצא האובייקט App, או במילים אחרות – היישום שלנו. אין צורך לדעת באופן מדויק באיזו תיקיה המשתמש בחר להתקין את היישום, אלא לפנות למאפיין Path של האובייקט App ולקבל את המידע הנדרש.

מכאן אפשר להבין שכדי לקבוע את תיקיית היישום כתיקיית העבודה, עלינו לכתוב את שורת הקוד הזו:

```
ChDir App.Path
```

שים לב, שהמשתמש יכול גם הוא לשנות את תיקיית העבודה באמצעות הפקד DirListBox. לחיצה אחת על אחת התיקיות המוצגות בפקד בוחרת בתיקיה, אך עדיין אינה משנה בפועל את תיקיית העבודה לתיקיה אחרת שנבחרה על ידי המשתמש. בחירת המשתמש מבוטאת במקרה זה במאפיין Path המקבל כערך את הבחירה. כאשר המשתמש לוחץ לחיצה כפולה הוא משנה בפועל את תיקיית העבודה.

הפקד FileListBox

הפקד **FileListBox** מציג את רשימת הקבצים שבכונן ובתיקיה הנוכחיים. כדי להגדיר לפקד מאיזו תיקיה אנו רוצים להציג את הקבצים, ניעזר במאפיין Path של הפקד. לדוגמה,

```
File1.Path = "C:\Windows"
```

אם נרצה לשלב בין שני הפקדים FileListBox ו-DirListBox בצורה כזו, שבחירת תיקיה בפקד DirListBox תגרום להצגת הקבצים של תיקיה זו ברשימת הקבצים שבפקד FileListBox, נכתוב את שורת הקוד הזו:

```
File1.Path = Dir1.Path
```

במקרה זה, כל הקבצים הקיימים בתיקיה הנבחרת יוצגו ברשימת הקבצים. אך מה נעשה כאשר נרצה להציג קבצים מסוג מסוים בלבד, כמו למשל הקבצים שהסיומת שלבם היא BAT, EXE ו- COM? לצורך כך נעזר במאפיין **Pattern** של הפקד. מאפיין זה קובע את סוגי הקבצים שיוצגו ברשימה. לדוגמה,

```
File1.Pattern = "*.exe; *.bat; *.com"
```

שים לב שוויז'ואל בייסיק תומכת בתו "**?**" כמייצג תו הכללה בשם קובץ. לדוגמה,

```
File1.Pattern = "A???.TXT"
```

במקרה זה, הקבצים שיוצגו בפקד FileListBox יהיו כל הקבצים שמתחילים באות A ובעלי 3 תווים נוספים ומסוג TXT בלבד.

מלבד בחירת קריטריון וסוג הקבצים שיוצגו בפקד ניתן לקבוע כקריטריון את מאפייני הקובץ (Attributes). ברירת המחדל היא שמוצגים כל הקבצים מלבד המוסתרים וקבצי מערכת, אולם, ניתן לשנות קביעה זו. הנה דוגמה לשורות קוד שגורמות להצגת קבצים לקריאה בלבד:

```
File1.ReadOnly = True
File1.Archive = False
File1.Normal = False
File1.System = False
File1.Hidden = False
```

כאשר המשתנה Normal מקבל ערך True, מוצגים כל הקבצים מלבד המוסתרים וקבצים לקריאה בלבד; כאשר Normal מקבל ערך False מוסתרים כל הקבצים. ניתן לקבוע את ערך קבצי הארכיון והקריאה בלבד ל- True ואז הם יוצגו.

מאפיין חשוב נוסף של הפקד FileListBox הוא **MultiSelect**. מאפיין זה מאפשר לבחור כמה קבצים יחד ובלעדיו – ניתן לבחור קובץ אחד בלבד. ברירת המחדל של המאפיין הוא הערך None - 0 כלומר, אין אפשרות לבחור בחירה מרובה, אלא קובץ אחד בלבד. הערכים 1 ו- 2 (Simple ו- Extended) מאפשרים בחירה מרובה. הערך 1 (Simple) מאפשר בחירה מרובה פשוטה. כלומר, לחיצה על שם קובץ בוחרת אותו, ולחיצה נוספת על שם זה מבטלת את הבחירה. לעומתו, הערך 2 (Extended) מאפשר בחירה באופן דומה לבחירת קבצים בסייר Windows.

המאפיין המקבל כערך את בחירת המשתמש (או במילים אחרות את שם הקובץ שנבחר מתוך הרשימה) הוא:

```
File1.Filename
```

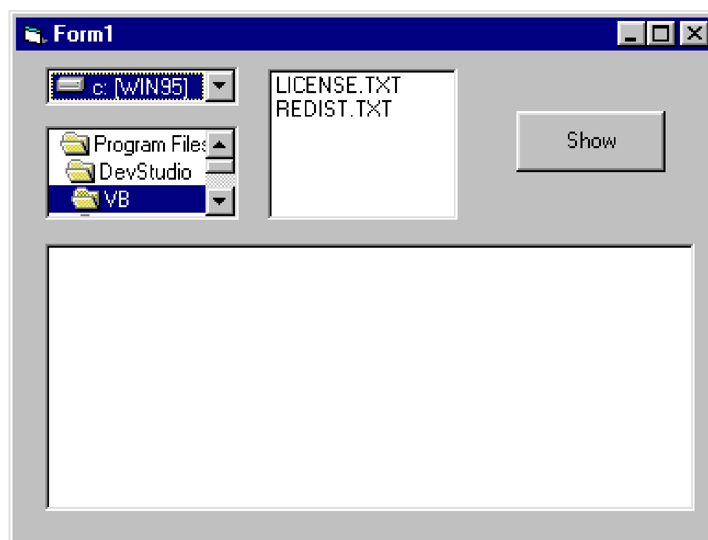
לדוגמה:

```
Dim SelectedFile As String
SelectedFile = File1.FileName
```

שילוב שלושת הפקדים

בדרך כלל, שלושת הפקדים למיפוי דיסק משתלבים יחד ביישום. בפני המשתמש ניתנת האפשרות לדפדוף ולשינוי הכונן, התיקיות והקבצים כאחד. נתבונן עתה כיצד ניתן לשלב יחד את שלושת הפקדים ממבט שורות הקוד.

כאשר המשתמש בוחר בכונן מסוים מתוך רשימת הכוננים, הוא מצפה שכל התיקיות הקיימות בכונן שבחר יוצגו בפקד רשימת התיקיות. באופן דומה, כאשר הוא בוחר בתיקיה מתוך רשימת התיקיות, הוא מצפה לראות ברשימת הקבצים את כל הקבצים, או חלק מהם אם נקבע להם קריטריון דרך המאפיין Pattern. כל שנותר הוא לבצע את הפעולה הרצויה, עיין בתרשים 11.14:



תרשים 11.14

בתרשים מוצג טופס המאפשר, בעזרת פקדי מיפוי דיסק, לבחור בדיסק בקובץ (מסוג TXT). בלחיצה על הלחצן Show יוצג תוכן הקובץ בתיבת הטקסט.

כיצד נבצע את הפעולות הללו בשורות קוד? בשלב הראשון נקבע את מאפייני הפקדים שבטופס על פי טבלה 11.4.

טבלה 11.4

פקד	מאפיין	ערך
DriveListBox	Name	drvMyDrive
DirListBox	Name	dirMyDir
FileListBox	Name	filMyFile
	Pattern	"*.txt"
CommandBottun	Name	cmdShow
TextBox	Name	txtFile
	Text	(Empty)
	MultiLine	True

בשלב שני ניגש לכתיבת הקוד לאירועים המתאימים, שיופעלו על ידי המשתמש.

כאשר המשתמש בוחר בכונן הרצוי מתוך רשימת הכוננים, הוא מפעיל את האירוע Change של הפקד. אם כן, האירוע Change הוא האירוע בו נרשום את שורות הקוד הגורמות לרשימת התיקיות להשתנות בהתאם לכונן הנבחר. הנה כך :

```
Private Sub drvMyPrive_Change()
        dirMyDir.Path = drvMyDrive.Drive
End Sub
```

זכור, אם המשתמש יבחר בכונן :A ולא יהיה בו דיסקט, התוכנית תופסק עם הודעת שגיאה. כדי למנוע זאת עלינו לכתוב מראש את שורות הקוד המטפלות במצבי שגיאה (נעשה זאת בפרק 20).

באירוע Change של הפקד drvMyDrive קבענו שהתיקיות שיוצגו בפקד dirMyDir יהיו אלו שנמצאות בכונן הנבחר. באופן דומה, בחירת המשתמש באחת מהתיקיות המוצגות ברשימה תפעיל את האירוע Change של הפקד dirMyDir, לכן נכתוב לאירוע זה את שורות הקוד המורות לפקד FilMyFile להציג את רשימת הקבצים (מסוג txt) הנמצאים בתיקיה שנבחרה.

שורות הקוד הן :

```
Private Sub dirMyDir_Change()
        FilMyFile.Path = dirMyDir.Path
End Sub
```

כאשר המשתמש בוחר בקובץ שברצונו להציג בתיבת הטקסט, אין צורך לכתוב לכך שורות קוד מיוחדות. המאפיין FilMyFile.Filename שומר את שם הקובץ הנבחר. כל שנותר הוא לכתוב את שורות הקוד לאירוע Click של הלחצן cmdShow אשר יפתח את הקובץ הנבחר ויעתיק את תוכנו אל תיבת הטקסט txtFile. שורות הקוד הדרושות :

```
Private Sub cmdShow_Click()
   Dim strSelectedFile As String
   Dim strTemp As String
   strSelectedFile = dirMyDir.Path & "\" & FilMyFile.FileName
   Open strSelectedFile For Input As #1
   Do While Not EOF(1)
      Line Input #1, strTemp
      txtFile.Text = txtFile.Text & strTemp
   Loop
   Close #1
End Sub
```

תיבות דו-שיח

אם המטרה שלנו להציג בפני המשתמש את היכולת הפשוטה של פתיחת קובץ או שמירתו, אין צורך להשתמש בפקדים למיפוי דיסק. ויז׳ואל בייסיק מספקת תיבות דו-שיח הדומות לאלו שבסביבת Windows, המאפשרות פתיחת קובץ, שמירתו, הגדרת מדפסת ומאפייני הדפסה וכו׳ .

בוויז׳ואל בייסיק קיימות חמש תיבות דו-שיח מסוג זה המקלות עלינו, כמתכנתים, את העבודה. במקום ליצור טופס מיוחד אשר יטפל בפתיחת קובץ, שמירתו, טיפול בהגדרת הדפסה וכו׳, נוכל להשתמש באחת מתיבות הדו-שיח המוכנות. אלו הן :

1. **Open** - לפתיחת קובץ.
2. **Save As** - לשמירת קובץ.
3. **Color** - לבחירת צבעים.
4. **Font** - לבחירת גופנים.
5. **Printer** - לבחירה והגדרה של הדפסה.

בתיבת דו-שיח נוספת, **Help**, לא נעסוק במסגרת ספר זה.

הפקד CommonDialog

כדי שנוכל להשתמש באחת מחמש תיבות הדו-שיח שמספקת לנו ויז׳ואל בייסיק, עלינו להוסיף פקד מיוחד לטופס. נבחר מארגז הכלים את הפקד **CommonDialog** ונמקם אותו במקום כלשהו בטופס.

אין חשיבות למיקום הפקד בטופס, מכיון שהוא אינו נראה למשתמש בזמן הפעלת התוכנית. אם פקד זה אינו מופיע בסרגל הכלים, נוסיף אותו באופן ידני (על כך למדנו בפרק 3).

ShowOpen

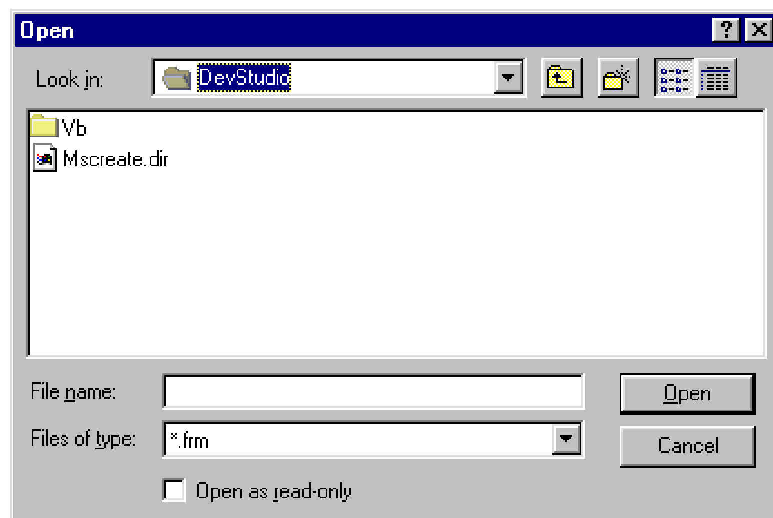
לאחר שמציבים את הפקד בטופס צריך להגדיר את שמו ואת סוג תיבת הדו-שיח שעליו לפתוח.

תחביר ההגדרה:

```
CommonDialogName.DialogType
```

בדוגמה הבאה, תיבת הדו-שיח שתופיע תאפשר לפתוח קובץ קיים (תרשים 11.15):

```
dlgMyDialog.ShowOpen
```



תרשים 11.15

שים לב ששם הפקד נקבע בחלון המאפיינים, ולא בשורות הקוד שבדוגמה. מעיון בתיבת הדו-שיח **Open** נראה שהמשתמש יכול לבחור קבצים מסוג frm אשר יופיעו בתיבת הדו-שיח ושאותם יוכל לפתוח. סוגי קבצים מוצגים בתיבת הגלילה "Files of type". את סוגי הקבצים ואת תיאורם ניתן לקבוע בעזרת המאפיין Filter, הנה כך:

```
description1\ filter1\ description2\ filter2
```

הערך description מייצג את הטקסט שיתאר את חתך הקבצים ו-filter מייצג את הקריטריון של הקבצים שיופיעו בפועל בתיבת הדו-שיח. לדוגמה,

```
dlgMyDialog.Filter = "Text Files (*.txt)| *.txt| _
                     All Files (*.*)| *.*"
dlgMyDialog.ShowOpen
```

את הקריטריון להצגת הקבצים צריך לכתוב לפני ההוראה להציג את תיבת הדו-שיח. כדי לקבוע את אחד מהקריטריונים כברירת מחדל, נשתמש במאפיין **Filter Index**.

לדוגמה:

```
dlgMyDialog.Filter = "Text Files (*.txt)| *.txt| _
                              All Files (*.*)| *.*"
dlgMyDialog.FilterIndex = 1
dlgMyDialog.ShowOpen
```

תיבת הדו-שיח מחזירה כערך את הקובץ שנבחר על ידי המשתמש. כאשר המשתמש לוחץ על הלחצן **Open** ערך זה מוחזר למאפיין **Filename**. כל שנותר הוא להשתמש במאפיין זה כדי לדעת איזה קובץ לפתוח. לדוגמה:

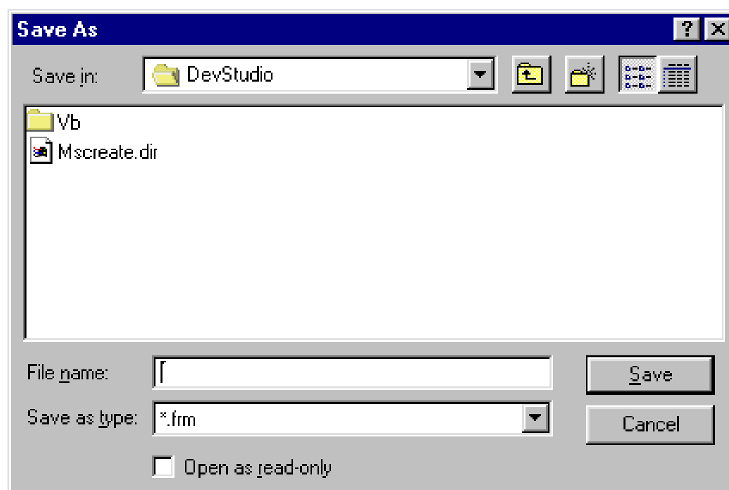
```
Open dlgMyDialog.FileName For Input As #1
```

כאשר המשתמש לוחץ על לחצן **Cancel**, מבטל את פעולת הבחירה וסוגר את תיבת הדו-שיח, היא אינה מחזירה ערך. כדי לטפל במצב בו המשתמש לוחץ על Cancel, נקבע את מאפיין התיבה **CancelError** ל-True. מאפיין זה קובע שכאשר המשתמש לוחץ על הלחצן Cancel הוא גורם לשגיאה. מכיון שכך, אנו כמתכנתים נדאג לטפל בשגיאה זו בקוד, או במילים אחרות, כאשר יש שגיאה בשלב זה של התוכנית נדע שהיא באה כתוצאה מבחירת המשתמש בלחצן Cancel. בשיגרת הטיפול בשגיאה נכתוב את שורות הקוד המתאימות, ובאופן טבעי נורה לתוכנית לסיים את השיגרה ולסגור את תיבת הדו-שיח. לדוגמה:

```
Private Sub mnuFileOpen_Click()
   On Error GoTo ErrHandler
   dlgMyDialog.ShowOpen
   Call OpenFile(dlgMyDialog.FileName)
   Exit Sub
ErrHandler:
   Exit Sub       'User Pressed Cancel button
End Sub
```

ShowSave

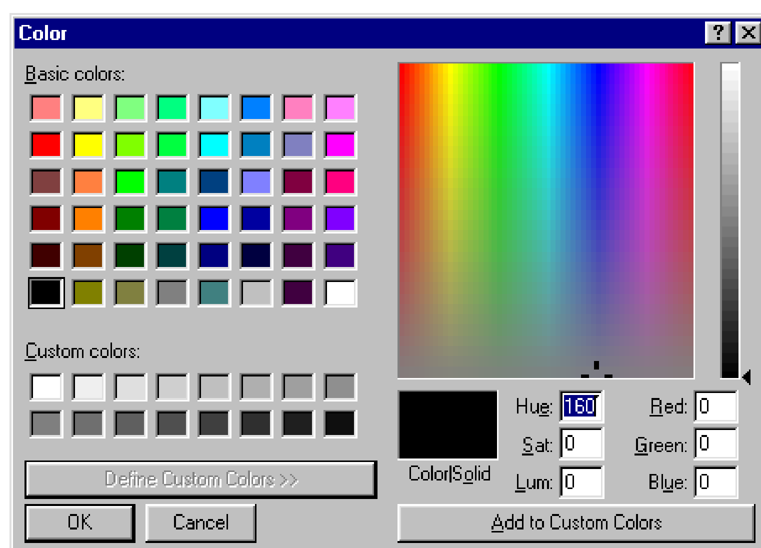
תיבת הדו-שיח **ShowSave** דומה לתיבת הדו-שיח ShowOpen במאפיינים ובערך שהיא מחזירה. ההבדל הוא בממשק המשתמש בלבד. במקום הכותרת **Open** והלחצן Open שבתיבת הדו-שיח ShowOpen, נמצא כאן את **Save As** ואת הלחצן **Save** בהתאמה (תרשים 11.16).



תרשים 11.16

ShowColor

תיבת הדו-שיח **ShowColor** מאפשרת למשתמש לבחור אחד מהצבעים הקיימים בלוח הצבעים, או ליצור צבע באופן ידני, על ידי ערבוב צבעים (תרשים 11.17).



תרשים 11.17

כאשר המשתמש בוחר בצבע הרצוי לו וסוגר את תיבת הדו-שיח, המאפיין **Color** של הפקד מקבל כערכך את הצבע שנבחר על ידי המשתמש. כדי לפתוח את לוח הצבעים

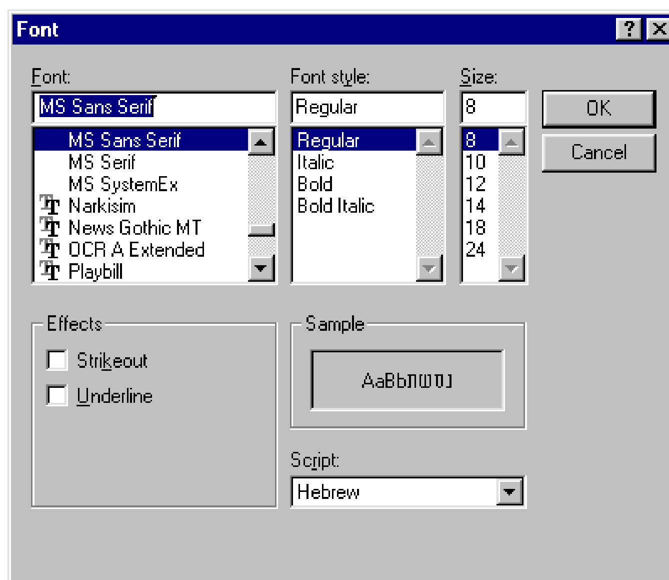
בהצגת הצבעים הבסיסיים של המערכת, צריך לאתחל את המאפיין **Flags** של הפקד עם הקבוע cdlCCRGBInit. לדוגמה:

```
Private Sub cmdColor_Click()
   On Error GoTo ErrHandler
   dlgMyDialog.CancelError = True
   dlgMyDialog.Flags = cdlCCRGBInit
   dlgMyDialog.ShowColor
   txtName.BackColor = dlgMyDialog.Color
   Exit Sub
ErrHandler:
   Exit Sub
End Sub
```

לחיצה על Cancel גורמת לשגיאה, המטופלת בשורת הקוד, בדיוק כמו שהדבר נעשה בתיבות הדו-שיח Open ו- Save.

ShowFont

תיבת הדו-שיח **ShowFont** מאפשרת למשתמש לבחור גופן, גודל וסגנון של כתב (תרשים 11.18).



תרשים 11.18

כאשר המשתמש לוחץ על הלחצן OK ומאשר את בחירתו, הוא מעדכן למעשה את המאפיינים הבאים של הפקד (טבלה 11.5):

טבלה 11.5

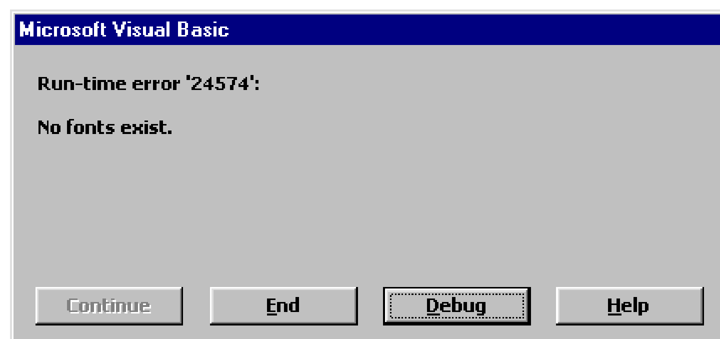
ערך	מאפיין
צבע הכתב	Color
שם גופן הנבחר	FontName
גודל הגופן	FontSize
גופן בולט	FontBold
גופן נטוי	FontItalic
קו תחתון	FontUnderline
קו חוצה	FontStrikethru

כדי שניתן יהיה להשתמש במאפיין **Color**, צריך להקדים ולהציב את הקבוע cdlCCRGBInit כערך במאפיין **Flags**. בעזרת המאפיין Flags גם ניתן לקבוע איזה סוגי גופנים יוצגו בתיבת הדו-שיח:

- הקבוע cdlCFScreenFonts גורם להצגת גופני מסך.
- הקבוע cdlCFPrinterFonts גורם להצגת גופני מדפסת.
- הקבוע cdiCFBoth מציג את שני הגופנים יחד.

זכור לקבוע את ערכי המאפיין Flags לפני הצגת תיבת הדו-שיח, אחרת תוצג הודעת שגיאה (תרשים 11.19).





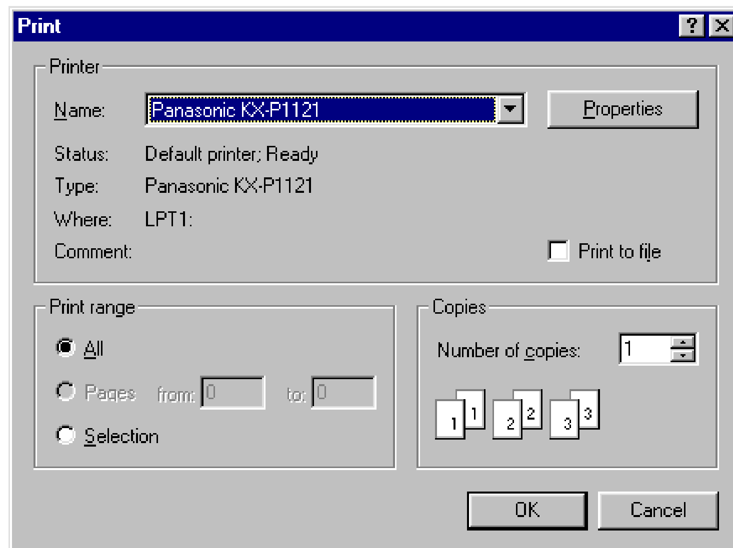
תרשים 11.19

שורות הקוד הבאות מדגימות שימוש בתיבת הדו-שיח **Show Font**, כדי לאפשר למשתמש לקבוע את הגופן וכל תכונות הכתב אשר יוצג בתיבת הטקסט txtName:

```
Private Sub cmdShow_Click()
   dlgFont.CancelError = True
   On Error GoTo ErrHandler
   dlgFont.Flags = cdlCFBoth or cdlCFEffects
   dlgFont.ShowFont
   txtName.Font.Name = dlgFont.FontName
   txtName.Font.Size = dlgFont.FontSize
   txtName.Font.Bold = dlgFont.FontBold
   txtName.Font.Italic = dlgFont.FontItalic
   txtName.Font.Underline = dlgFont.FontUnderLine
   txtName.Font.Strikethru = dlgFont.FontStrikethru
   txtName.ForeColor = dlgFont.Color
   Exit Sub
ErrHandler:
   Exit Sub
End Sub
```

ShowPrinter

תיבת הדו-שיח **Show Printer** מאפשרת למשתמש להגדיר מדפסת רצויה ואת הגדרות הדפסה של הפלט המיועד (תרשים 11.20).



תרשים 11.20

באמצעות תיבת דו-שיח זו המשתמש יכול לשנות את הגדרות המדפסת ולשנות את מדפסת ברירת המחדל. בנוסף, המשתמש קובע את מספר העותקים הנשלחים להדפסה, אילו דפים מתוך המסמך להדפיס ועוד.

ברגע שהמשתמש לוחץ על הלחצן OK ומאשר את ההגדרות שבתיבת הדו-שיח, הוא מעדכן את המאפיינים הבאים של הפקד (טבלה 11.6):

טבלה 11.6

משמעות	מאפיין
מספר עותקים	Copies
מספר הדף שממנו תתחיל ההדפסה	FromPage
הדף האחרון בהדפסה	ToPage

שליחת הפלט למדפסת

בעזרת תיבת הדו-שיח **ShowPrinter** המשתמש קובע את הגדרות ההדפסה ואת אופן ההדפסה, אך אין הוא שולח בפועל את הפלט למדפסת. שליחת הפלט למדפסת נעשית בעזרת **Printers Collection** (אוסף המדפסות). זהו אובייקט המכיל את כל המדפסות המעודכנות (והניתנות לשימוש) במערכת ההפעלה.

את אוסף המדפסות מייצג האובייקט **Printer**. טבלה 11.7 מציגה חלק מהמאפיינים והשירותים של אובייקט זה ואת משמעותם:

טבלה 11.7

משמעות	מאפיין
מיקום הפלט בקואורדינטת X בדף	CurrentX
מיקום הפלט בקואורדינטת Y בדף	CurrentY
רוחב הדפסה בדף	ScaleWidth
אורך הדפסה בדף	ScaleHight
מס' עותקים	Copies
	שירות
רוחב הטקסט בדף	TextWidth (String)
אורך הטקסט בדף	TextHight (String)
הדפס	Print
דף חדש	NewPage
מעדכן הגדרות ושולח את המסמך להדפסה **בפועל**	EndDoc
מפסיק **מיידית** את ביצוע ההדפסה	KillDoc

שורות הקוד הבאות מדגימות הדפסה של המשפט "Document Title" במרכז הדף (בשורה העליונה):

```
Dim strMsg As String
Dim varMsgWidth

strMsg = "Document Title"
varMsgWidth = Printer.TextWidth(strMsg) / 2
Printer.CurrentX = Printer.ScaleWidth / 2 - varMsgWidth
Printer.Print strMsg
Printer.EndDoc
```

רשימת המדפסות באוסף מקבילה לרשימת המדפסות המופיעה בקטגוריה **מדפסות** (Printers) שבחלון לוח הבקרה. לכל מדפסת יש אינדקס המזהה אותה (כאשר המדפסת הראשונה בעלת ערך 0). האובייקט Printer פונה תמיד ובאופן אוטומטי למדפסת המוגדרת כמדפסת ברירת המחדל. אבל יש אפשרות לבחור מדפסת אחרת מהאוסף ולהגדיר אותה לצורך ההדפסה, על פי התחביר הבא:

```
Set Printer = Printers(n)
```

כאשר n מייצג את האינדקס של המדפסת שנבחרה מתוך האוסף.

משפט זה משנה את מדפסת ברירת המחדל בלבד. אין אפשרות להוסיף או למחוק מדפסות בעזרת האובייקט Printer. פעולות אלו מתבצעות דרך לוח הבקרה בלבד. אחרי שנבחרה מדפסת, ניתן להדפיס ישירות אליה.

האובייקט Printer מספק מיגוון רחב של שירותי הדפסה של טקסט וגרפיקה, למידע נוסף היעזר במערכת העזרה של ויז'ואל בייסיק.

הדפסת טופס

ניתן להדפיס טופס ישירות למדפסת באמצעות השירות **PrintForm**. לדוגמה:

```
FrmMain.PrintForm
Me.PrintForm
```

כאשר אין מציינים את שם הטופס, ויז'ואל בייסיק מדפיסה את הטופס הנוכחי. אם אנו לא מעוניינים להדפיס את כל מה שמוצג בטופס, כמו למשל חלק מהפקדים המצויים בו, יש להגדיר תחילה את המאפיין Visible שלהם ל-False ורק אחר כך לקרוא לשירות PrintForm.

תרגול

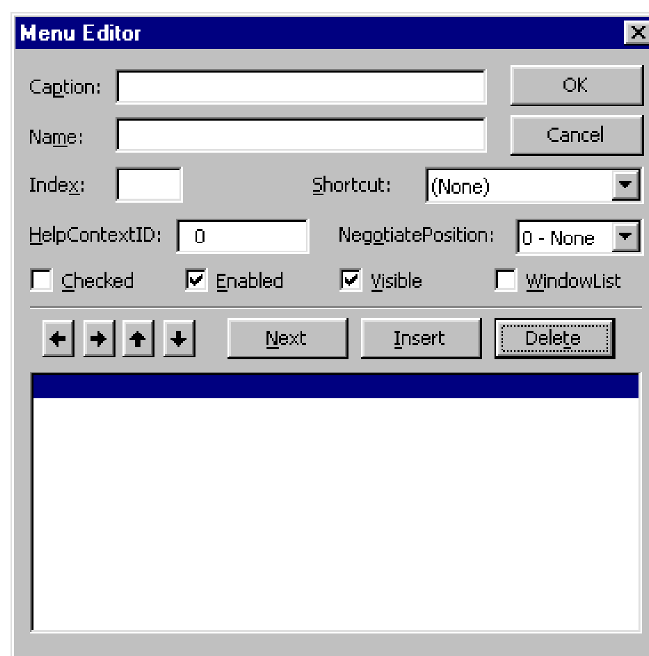
1. כתוב תוכנית אשר תקבל שם קובץ ומסלול, ותדפיס את תוכנו לתוך תיבת טקסט.
2. בצע את תרגיל 1, אך הפעם במקום שהמשתמש יקיש שם קובץ, פתח תיבת דו-שיח ומתוכה הוא יבחר בקובץ המתאים. שים לב שעל תיבת הדו-שיח להציג קבצי txt בלבד.

12: תפריטים

יישומים רבים בסביבת העבודה החלונאית משלבים בתוכם תפריטים. התפריטים, מלבד הנוחות והבהירות שהם מקנים ליישום, חוסכים בו מקום רב. בוויז'ואל בייסיק קיים **עורך תפריטים**, בעזרתו תוכל להוסיף שורת תפריטים ליישום שלך.

עורך התפריטים

כדי להציג את עורך התפריטים (תרשים 12.1) בחר באפשרות **Menu Editor** שבתפריט **Tools**, או לחץ על הלחצן שבסרגל הכלים (לחילופין, הקשה על Ctrl+E מציגה את העורך).



תרשים 12.1

בעזרת עורך התפריטים תוכל לבנות את שורת התפריטים. בנוסף, ניתן לקבוע באמצעות העורך את המאפיינים העיקריים של כל תפריט ותפריט. את מאפייני

התפריט ניתן לשנות באופן דינמי בזמן הפעלת התוכנית, ובכלל זה הוספה דינמית של אפשרויות לתפריט.

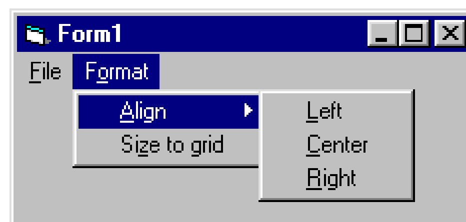
בניית ההיררכיה

שורת התפריטים מציגה למשתמש את התפריטים הראשיים בלבד, בשעה ששאר האפשרויות אינן גלויות. ניקח לדוגמה את התפריט **File** (המצוי כמעט בכל יישום), כל האפשרויות שבו מוסתרות והמשתמש רואה את שם התפריט **File** בלבד. רק כאשר המשתמש בוחר בתפריט מוצגת בפניו רשימת האפשרויות שבתפריט (תרשים 12.2). אפשרויות התפריט הן תת-תפריטים של התפריט הראשי File, והן נמצאות תחתיו בהיררכיה.



תרשים 12.2

כאשר נבנית ההיררכיה, התפריט File נמצא בראשה והאפשרויות שכלולות בו נמצאות רמה אחת מתחתיו. תת-תפריט של אחת מאפשרויות אלו יהיה רמה נוספת למטה (תרשים 12.3).

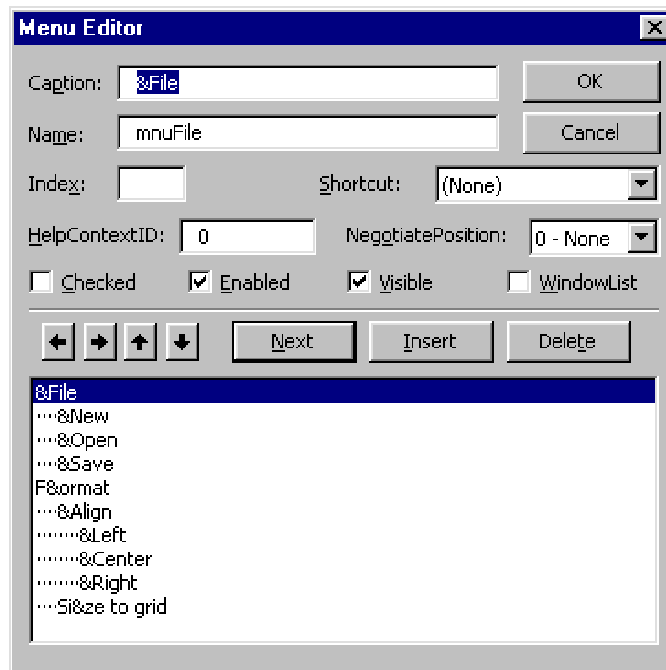


תרשים 12.3

שים לב שהתפריט **Format** בתרשים 12.3 נמצא באותה רמה של התפריט File, כי שניהם תפריטים ראשיים.

חלון עורך התפריטים מחולק לשניים (תרשים 12.1): בחלקו העליון ניתן להגדיר את המאפיינים ובחלקו התחתון נבנית ההיררכיה. כל רמה בהיררכיה באה לידי ביטוי בתזוזה ימינה של האפשרות (תרשים 12.4). ככל שהאפשרות נמוכה יותר מבחינה היררכית, היא מוצגת ימינה יותר (זוהי מערכת תפריטים באנגלית, ולכן התצוגה

נפרשת ימינה). התפריטים השמאליים ביותר הם התפריטים הנמצאים בראש ההיררכיה והם גלויים מייד למשתמש.



תרשים 12.4

בראש החלק התחתון של חלון העורך יש שלושה לחצנים: Next, Insert ו- Delete.

בעזרת הלחצן **Delete** ניתן למחוק את האפשרות המסומנת בתיבת הרשימה, כאשר נרצה לבטל תפריט (או אפשרות) משורת התפריטים.

הלחצן **Insert** גורם להכנסת תפריט חדש לפני ("מעל") התפריט (או האפשרות) המסומנת בתיבת הרשימה.

הלחצן **Next** מוסיף פריט נוסף לרשימה.

שים לב שניתן לעלות ולרדת ברשימה בעזרת לחצני החיצים מעלה-מטה בהתאמה.

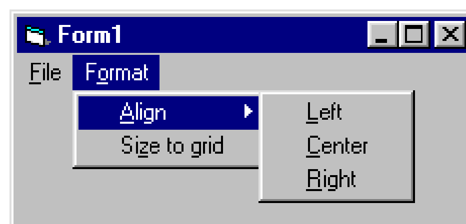
תת-תפריט

כדי להפוך תפריט ברשימה לתת-תפריט (או לאפשרות בתפריט ראשי) נמקם אותו מעט ימינה ברשימה. "דחיפה" ימינה של התפריט ממקמת אותו רמה אחת למטה, לחילופין, "דחיפה" שמאלה ממקמת אותו רמה אחת למעלה.

שים לב, התפריטים הראשיים (המרכיבים את שורת התפריטים נמצאים בצד השמאלי ביותר של הרשימה). "דחיפת" תפריטים ימינה ושמאלה נעשית בעזרת לחצני החיצים ימינה ושמאלה בהתאמה.

בתרשים 12.4 מופיע התפריט File בצד השמאלי ביותר. תפריט זה נמצא בראש ההיררכיה והוא התפריט שיופיע בשורת התפריטים שביישום. תחתיו נמצאים תת-התפריטים (האפשרויות) New, Open ו-Save. תפריטים אלה נמצאים תחת התפריט File ומעט ימינה ביחס למיקומו. מיקום זה מגדיר אותם כתת-תפריטים של התפריט File.

התפריט **Format**, למשל, נמצא במיקום זהה לתפריט File, ולכן הוא יוצג כתפריט ראשי ויופיע בשורת התפריטים שביישום. לחיצה על הלחצן הימני גורמת ליצירת תת-תפריט של התפריט Format, זו האפשרות **Align**. לחיצה נוספת על הלחצן הימני יוצרת תת-תפריט לתת-התפריט Align - האפשרויות Left, Center ו- Right. במקרה זה, יופיע חץ לצד התפריט Align המורה שבתפריט זה נמצאות אפשרויות נוספות (תרשים 12.5).



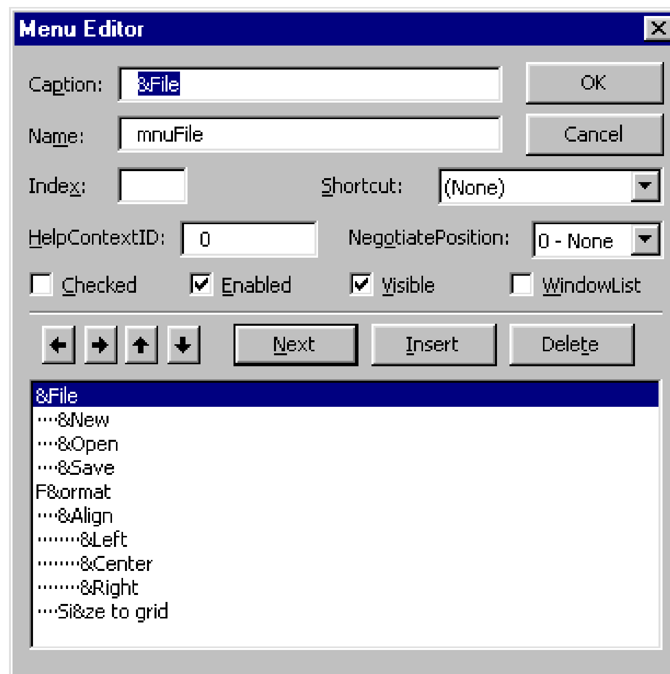
תרשים 12.5

מאפיינים

בחלק העליון של חלון עורך התפריטים נקבעים המאפיינים של כל תפריט והאפשרויות המרכיבות את שורת התפריטים. חלק ממאפיינים אלה אפשר לקבוע באופן דינמי משורות הקוד שבתוכנית.

המאפיין הראשון **Caption** קובע את כותרת התפריט. כותרת זו רואה המשתמש בשורת התפריטים של היישום, כדוגמת File, Edit, Format וכו'. מאפיין זה גם מאפשר לקבוע את "**מקש הגישה**" לתפריט. "מקש גישה" הינו המקש במקלדת שמקשים עליו בצירוף המקש Alt כדי לבחור בתפריט ללא שימוש בעכבר. מקש גישה מאופיין בקו תחתי תחת האות בשם התפריט, המייצגת אותו. לדוגמה, בתפריט "File" לחיצה על Alt+F במקלדת כמוה כבחירה בתפריט File בעזרת העכבר.

כדי לקבוע את מקש הגישה לתפריט, צרף את התו & (במאפיין Caption) לפני האות אשר קבעת כמקש גישה לתפריט. לדוגמה, כדי ליצור את התפריט "File" (שבו מקש הגישה Alt+F) קבע את ערך המאפיין Caption ל- "&File" (תרשים 12.6).



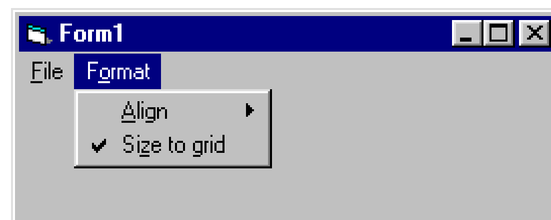
תרשים 12.6

צירוף התו & לפני האות F יגרום להצגתו בתפריט עם קו תחתי (F) ולהפיכתו למקש גישה. הקשה על Alt+F במקלדת מייצגת בחירה בתפריט File.

המאפיין **Name** משמש בתפקיד זהה למאפיין Name של פקד רגיל. בשם זה מתייחסים לתפריט או לאחת מאפשרויותיו בשורות הקוד.

המאפיין **Index** קובע את ערך האינדקס של התפריט, אם תפריט זה הוא חלק ממערך תפריטים. אם לא קיים מערך תפריטים, אין למאפיין זה כל משמעות. כאשר כמה תפריטים מהווים מערך, לכולם יהיה אותו שם (Name) והם יובדלו בערך האינדקס שלהם, כאשר האיבר הראשון במערך יהיה בעל ערך אינדקס אפס.

המאפיין **Checked** גורם לסימון (או לביטול הסימון) ✓ לצד התפריט (תרשים 12.7).

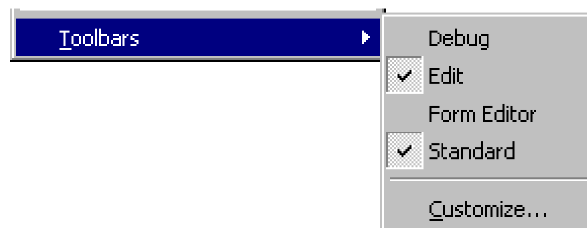


תרשים 12.7

בתפריט יש אפשרויות אשר הציון לכך שהן נבחרו הוא הסימון ✓ לצידן. בחירה נוספת באפשרות מוחקת את הסימון ✓ ומבטלת את פעולתן. הוספה והסרה של הסימון ✓ מהתפריט נעשים בעזרת המאפיין **Checked**. באופן דינמי המאפיין יקבל ערך True להוספת הסימון ✓ וערך False להסרתו. לדוגמה,

```
mnuFile.Checked = True
```

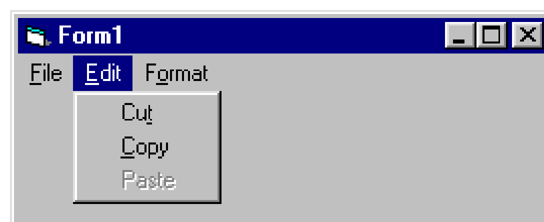
דוגמה נפוצה לשימוש במאפיין זה היא האפשרות Toolbars שהוא תת-תפריט לתפריט View (תרשים 12.8).



תרשים 12.8

בחירה בסרגל הרצוי (כדוגמת Edit ו- Standard בתרשים 12.8) מציגה סרגלי כלים אלה. ביטול הבחירה, תבטל את הסימון ✓ ותסתיר את הסרגל.

המאפיין **Enable** קובע אם התפריט יהיה פעיל או לא. תפריט לא פעיל מוצג באפור ולחיצה עליו אינה מאפשרת את הפעלתו (תרשים 12.9). כלומר, סימון ✓ בתיבת הסימון של המאפיין Enable גורם לתפריט להיות פעיל וביטול הבחירה גורם לתפריט להיות בלתי פעיל.



תרשים 12.9

דוגמה מוכרת מסביבת חלונות לתופעה בה אחת מהאפשרויות בתפריט אינה פעילה, היא האפשרות **Paste** שבתפריט **Edit**. כל עוד לא בוצעה העתקה (Copy) או גזירה (Cut) ללוח, אין משמעות לאפשרות ההדבקה (Paste). במצב זה האפשרות Paste אינה פעילה. כאשר המשתמש מבצע את אחת האפשרויות Copy או Cut הופכת האפשרות Paste לפעילה. הפיכת תפריט לפעיל או ללא פעיל נעשית בעזרת המאפיין Enable.

המאפיין **Visible** קובע אם התפריט ייראה למשתמש או לא. סימון ✓ בתיבת הסימון Visible גורם להצגת התפריט (או אחת מאפשרויותיו). ביטול הבחירה גורם להסתרת

התפריט מעיני המשתמש. כאשר התפריט מוסתר, מקשי הגישה והקיצור שלו אינם פעילים.

ביצוע אפשרות התפריט בשורות קוד

בדומה לפקד רגיל היודע להגיב לאירועים שונים, גם לחיצה על התפריט גורמת להתרחשות האירוע Click של התפריט. האירוע Click של התפריט הוא השיגרה בה צריך לכתוב את שורות הקוד שיתבצעו בזמן בחירה בתפריט. לדוגמה,

```
Private Sub mnuFileExit_Click()
   End
End Sub
```

שיגרה זו תופעל כאשר המשתמש בוחר באפשרות Exit שבתפריט File. במקרה זה התוכנית תבצע את המשפט End והיישום ייסגר. והנה דוגמה נוספת:

```
Private Sub mnuFileClose_Click()
   Unload Me
End Sub
```

השיגרה המופעלת כתוצאה מבחירת המשתמש באפשרות Close שבתפריט File, סוגרת את הטופס הנוכחי ומשחררת אותו מהזיכרון.

הוספת תפריטים באופן דינמי

במהלך ביצוע התוכנית צריך לפעמים להוסיף באופן דינמי אפשרויות תפריט שלא היו קיימות בתפריט בתחילה, או שהוסתרו. כאשר האפשרויות מוגדרות מראש, ניתן ליצור אותן בעורך התפריטים ולהיעזר במאפיין **Visible** כדי להציגן בזמן המתאים. פתרון זה אינו יעיל כאשר האפשרויות אינן קבועות או אינן ידועות מראש.

דוגמה נפוצה לשימוש בהוספה דינמית של אפשרויות לתפריט היא רשימת המסמכים האחרונים שנפתחו ב- Word (תרשים 12.10). במקרה זה, שמות הקבצים אינם ידועים מראש ויש להוסיפם באופן דינמי בלבד.



תרשים 12.10

כדי להוסיף אפשרות באופן דינמי לתפריט, היא חייבת להיות חלק ממערך. את האיבר הראשון במערך יוצרים מראש בעורך התפריטים. איבר זה יקבל את ערך האינדקס 0 (אפס) וכל איבר נוסף שנוסף באופן דינמי יקבל את הערך הבא לפי הסדר. אם לא תרצה שהאיבר הראשון במערך יוצג למשתמש, הסתר אותו בעזרת המאפיין Visible, אך קיומו במערך האפשרויות הכרחי לצורך הוספת אפשרויות באופן דינמי.

כדי להוסיף איבר (אפשרות) למערך צריך להשתמש במשפט **Load**. לדוגמה,

```
Load mnuFileRecent(1)
```

משפט זה מוסיף איבר למערך (המתבטא כאפשרות נוספת בתפריט). כדי להוסיף אפשרות נוספת השתמש שוב במשפט Load, אך הפעם ערך האינדקס יגדל ל- 2. לדוגמה,

```
Load mnuFileRecent(2)
```

עתה, כשהאפשרויות נוצרו, נשאר רק לקבוע את ערכי המאפיינים שלהן. לדוגמה:

```
mnuFileRecent(1).Visible = True
mnuFileRecent(1).Caption = "C:\My Documents\Vb12.doc"
mnuFileRecent(2).Visible = True
mnuFileRecent(2).Caption = strFileName
mnuFileRecent(2).Checked = True
```

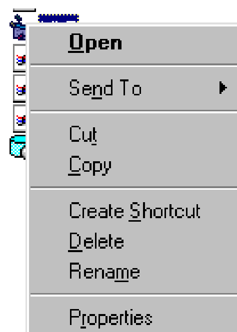
כדי להסתיר אפשרות מהתפריט, השתמש בשירות **Hide**, או קבע את ערך המאפיין Visible ל- False.

אפשרות מוסתרת עדיין קיימת בזיכרון. כדי למחוק אותה ולשחרר את הזיכרון השתמש במשפט **Unload**. לדוגמה,

```
Unload mnuFileRecent(2)
```

Popup Menu

Popup Menu הוא **התפריט המקוצר** ("תפריט קופץ") המוכר מיישומים הפועלים בסביבת חלונות ומתקבל בלחיצה ימנית על העכבר (תרשים 12.11).



תרשים 12.11

Popup Menu מכיל לפחות אפשרות אחת ואין הוא כולל כותרת כמו בשורת התפריטים הרגילה. כדי להציג תפריט מקוצר באופן דינמי במהלך התוכנית, מספקת לנו ויז'ואל בייסיק את השירות PopupMenu.

תחביר השירות הוא:

```
[Object].PopupMenu menuname [,flags,x,y,boldcommand]
```

הארגומנטים flags, x, y ו-boldcommand אופציונליים ואין חובה לשלוח אותם לשירות.

משמעות הארגומנטים:

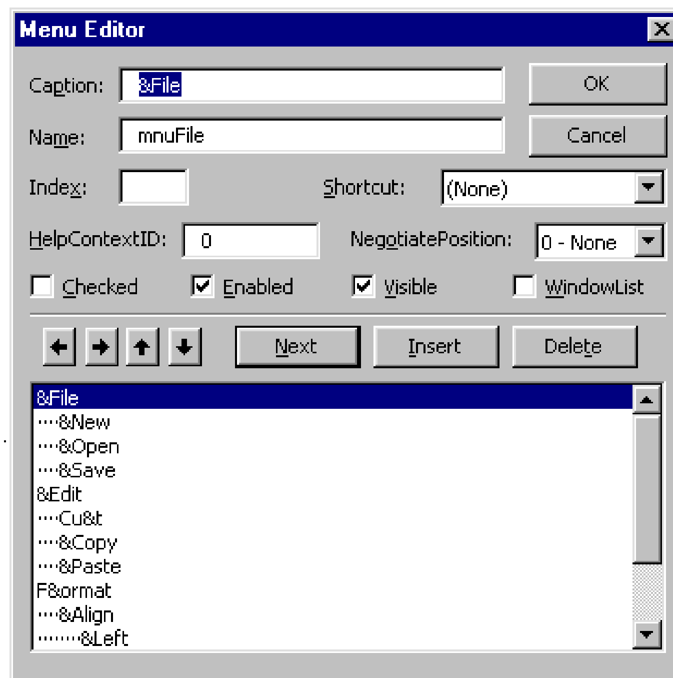
- **menuname** - שם התפריט (במערך התפריטים) שאת אפשרויותיו יציג התפריט המקוצר. תפריט זה יוצג גם בשורת התפריטים, אלא אם נבטל את הצגתו בעזרת המאפיין Visible.
- **flags** - קובע את מקומו של התפריט המקוצר. לארגומנט זה שלושה קבועים:

1. **vbPopupMenuLeftAlign** - קובע את מיקום התפריט משמאל לקואורדינטה X בה נלחץ העכבר.
2. **vbPopupMenuCenterAlign** - מציב את התפריט במרכז ביחס לקואורדינטה X בה נלחץ העכבר.
3. **vbPopupMenuRightAlign** - מציב את התפריט מימין לקואורדינטה X בה נלחץ העכבר (ערך זה הוא ערך ברירת המחדל).

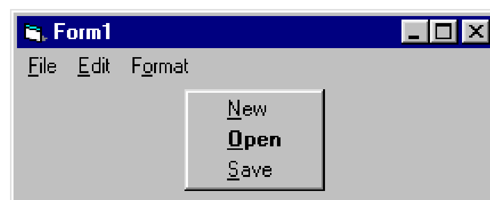
- **x,y** - קובעים באופן מדויק את מיקום התפריט המקוצר בטופס.
- **boldcommand** - קובע איזו אפשרות מבין האפשרויות תוצג באופן בולט.

בהתייחס לחלון עורך התפריטים המוצג בתרשים 12.12, שורת הקוד הבאה תציג את התפריט המקוצר המודגם בתרשים 12.13.

```
Me.PopupMenu mnuFile, , , , mnuFileOpen
```



תרשים 12.12



תרשים 12.13

תרגול

1. צור את הטופס הבא:

לחיצה על האפשרות Font תגרום להצגת תיבת דו-שיח המציגה גופנים. בחירת המשתמש תשנה את הכתוב בתיבת הטקסט לגופן שנבחר. כך גם לגבי הגודל.

2. צור את הטופס הבא:

לחיצה על הלחצן **הוסף** תגרום להוספת אפשרות לתפריט ששמו נכתב בתיבת הטקסט.

חלק ב'

תכנות מקצועי

13 : מבוא ל- ActiveX

ActiveX DLL

כשמריצים את תוכנת ויז'ואל בייסיק או פותחים פרויקט חדש, מוצג לפנינו חלון בו ניתן לבחור באיזה סוג פרויקט נרצה ליצור. אחת האפשרויות היא ActiveXDLL. פרויקט מסוג זה מאפשר לנו לבנות ActiveX שיהודר אחר כך לקובץ dll.

מדוע ליצור ActiveX או DLL?

השימוש ב-ActiveX משפר את יכולת הפיתוח שלנו כמתכנתים. לפעמים נרצה לפתח בכלים אשר מיגוון האפשרויות התכנותיות אשר הם מספקים מוגבלות. במקרה זה ActiveX (או בצורתו כקובץ DLL) יעזור לנו מאוד. את כל היכולות והאפשרויות שנרצה שהתוכנות תבצענה ניצור ב-ActiveX ואותו נקשר אל כלי הפיתוח (המוגבל) בו נפתח. העבודה מול ActiveX תיעשה על ידי שימוש במאפיינים אותם הוא מספק ובקריאה לשירותים שלו.

שים לב, שכל כלי פיתוח התומך במיכון (Automation) מספק לנו כלים לעבודה מול ActiveX.

שימוש נוסף שנעשה כיום באמצעות ActiveX ואשר מקבל תאוצה, הוא הפיתוח בארכיטקטורת Tiers - 3. ארכיטקטורה זו קובעת כי כל יישום יפותח בשלוש רמות (או שכבות). השכבה העליונה היא שכבת היישום עמו עובד משתמש הקצה. השכבה התחתונה היא שכבת השרתים (Servers) כדוגמת שרת מסד הנתונים וכדו', המספקים שירותים שונים ליישום. ביניהן מפרידה שכבת הביניים בה יושב ActiveX ואשר תפקידו לתווך בין שתי השכבות העליונה והתחתונה. כאשר ליישום דרוש מידע הנמצא במסד הנתונים הוא מפנה את בקשתו ל-ActiveX וזה מבצע עבורו את השאילתה ממסד הנתונים ומחזיר את התשובה חזרה ליישום.

היתרון שבשימוש בארכיטקטורה זו הוא, שלשכבת היישום אין תלות בשכבה התחתונה המספקת לו שירותים. היות שכך, כאשר מתחלפת השכבה התחתונה או משתנה (למשל הוחלף מסד הנתונים), אין צורך לבצע שינויים ביישום עצמו, פשוט נעדכן את ActiveX, היושב באמצע, באופן שיתאים לשינוי בשכבה התחתונה.

בחלקו הראשון של הפרק נלמד כיצד ליצור פרויקט מסוג ActiveXDLL ולהדרו לקובץ dll המספק שירותים שונים או מהווה שכבה שנייה במודל Tiers - 3.

הפעל את ויז'ואל בייסיק או צור פרויקט חדש ובחר באפשרות ActiveXDLL. בניגוד לפרויקט סטנדרטי (StandardEXE), אשר יוצר כברירת מחדל את הטופס הראשון, בפרויקט ActiveXDLL נוצרת מחלקה חדשה.

שים לב, ניתן להוסיף לפרויקט מסוג ActiveXDLL טופס רגיל במידה ו-ActiveX עושה בו שימוש. אולם, ActiveX לא חייב לכלול טפסים כלל. לחילופין, פרויקט רגיל יכול לכלול מחלקות אף הוא.

מחלקה (Class Module)

מחלקה בוויז'ואל בייסיק מיוצגת על ידי קובץ מסוג cls. קובץ זה הוא אחד מהקבצים המרכיבים את הפרויקט כולו.

מחלקה יכולה לכלול משתנים (Data Members), מאפיינים (Properties), שירותים (Methods) ואירועים (Events). המחלקה שומרת את הנתונים בתוך המשתנים ומספקת מאפיינים ושירותים חיצוניים לכלים אשר ישתמשו במחלקה זו.

שים לב, למרות שמודול (Module) רגיל מספק פונקציות ושגרות ציבוריות אשר ניתן לפנות אליהן, עדיין הוא שונה במהותו ממחלקה. במחלקה, בניגוד למודול, אנו יכולים ליצור מספר מופעים (Instances) של אותה מחלקה ואף מכלי פיתוח (או פרויקטים) החיצוניים ל-dll אליו שייכת המחלקה.

משתנים (Data Members)

המידע, בו משתמשת המחלקה, נשמר במשתני המחלקה. בהתאם לכללי תכנות מוכוון אובייקטים (Object Oriented) משתנים אלו יוגדרו כ-Private. במצב זה לא ניתן לפנות אליהם או לשנות את ערכם מחוץ למחלקה עצמה, כך נשמרת ההגנתיות (Incapsulation) על משתנים אלו, מכיון שרק מאפיין או שירות של המחלקה עצמה יכול לבצע שינויים במשתנים אלו. אם הצהרת על אחד המשתנים כ-Public, ניתן יהיה לשנות את ערכו גם שלא דרך המאפיינים והשירותים של המחלקה.

ההצהרה על משתני המחלקה נעשית כהצהרה רגילה על משתנים בוויז'ואל בייסיק.

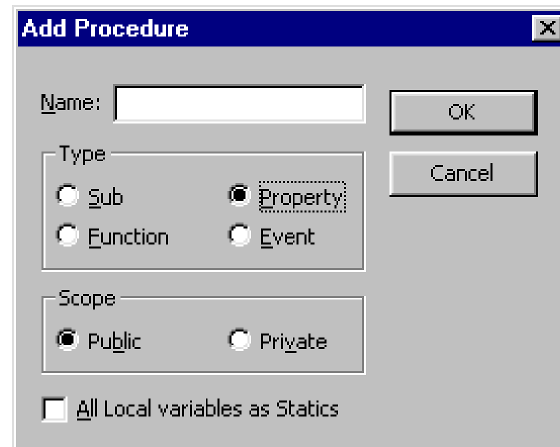
לדוגמה:

```
Private Fname As String
Private Age As Integer
```

מאפיינים (Properties)

באפשרותך להוסיף משתנים למחלקה, ואינך מוגבל במספר המאפיינים אותם תוכל להוסיף. מטרת המאפיינים היא לאפשר גישה לנתונים לצורך קריאה ועדכון ערכיהם. מכיון שבדרך כלל יוצהרו משתני המחלקה כ-Private, נעזר במאפיינים לצורך עדכון או הצגת ערכי משתני המחלקה.

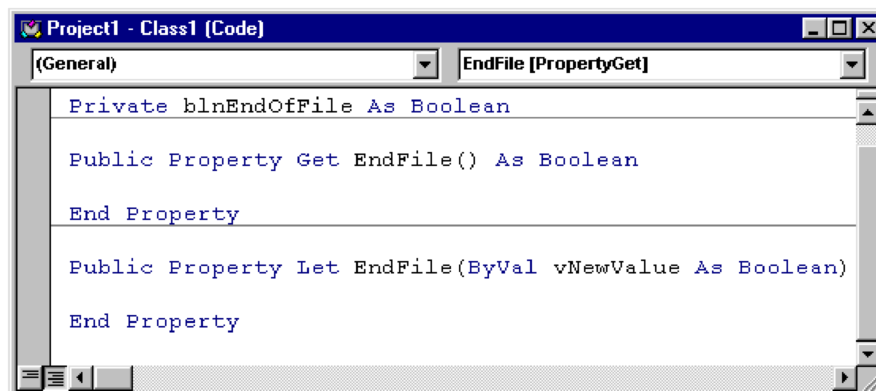
כדי להוסיף מאפיין למחלקה, בחר באפשרות **Add Procedure** שבתפריט **Tools**. בחירה באפשרות זו תציג את החלון **Add Procedure** (תרשים 13.1) בו תסמן את האפשרויות **Property** ו- **Public**.



תרשים 13.1

הקלד לתיבת הטקסט Name את שם המאפיין ואשר את הפעולה.

ויז'ואל בייסיק יוצרת אוטומטית מאפיין בשם זה המבצע שתי פעולות. האחת, **Get**, מאפשרת קריאת ערכי משתנה (או משתנים) של המחלקה, והשנייה, **Let**, מאפשרת לקבוע ערכים למשתנה זה (תרשים 13.2). שים לב, כי בהתאם לתפקידם, Let מקבל ערך ואילו Get מחזיר ערך. הערכים שמקבל או מחזיר הפונקציה יכולים להשתנות על ידך בהתאם לסוג המשתנים שהגדרת.



תרשים 13.2

לפי זה למשתנה **blnEndOfFile** של המחלקה **MyFile** נוכל לפנות בשתי דרכים.

האחת:

```
Dim clsFile As New MyFile
Dim blnEof As Boolean

blnEof = clsFile.EndFile
```

שורת קוד זו נעזרת במאפיין **Get EndFile** של המחלקה MyFile.

ואילו שורות הקוד הבאות ייעשו שימוש במאפיין **Let EndFile** כדי לעדכן את משתנה המחלקה:

```
Dim clsFile As New MyFile
Dim blnEof As Boolean

clsFile.EndFile = blnEof
```

שירותים (Methods)

מלבד מאפיינים, מספקת המחלקה שירותים שונים. שירותים אלו הם שגרות ופונקציות לכל דבר. כאשר השירות מחזיר ערך נשתמש בפונקציה, וכאשר הוא מבצע פעולה כלשהי ללא החזרת ערך, נכתוב אותו כשיגרה.

לדוגמה, השירות **BigLetters** מקבל כפרמטר כתובת של משתנה מסוג מחרוזת (String) והופך את אותיות המחרוזת אותה הוא מקבל לאותיות גדולות. השירות ייראה כך:

```
Public Sub BigLetters(strWord As String)
      strWord = UCase(strWord)
End Sub
```

שים לב, כדי שנוכל לקרוא לשירות הוא חייב להיות מוצהר כ-Public.

הקריאה לשירות ממקור חיצוני למחלקה ייראה כך:

```
Dim clsEmp As New Employees
Dim strName As String

strName = "Danny"

clsEmp.BigLetters strName

Print strName
```

התוצאה שתודפס תהיה DANNY.

אם השירות מחזיר ערך נבנה אותו כפונקציה.

שים לב, השירותים הם בעצם פונקציות ושגרות רגילות אשר חלים עליהם כל הכללים עליהם למדת בפרק העוסק בשגרות ופונקציות. כמו כן, ניתן להוסיף למחלקה

אירועים (Events) להם תדע המחלקה להגיב, אולם לא נעסוק באירועים במסגרת מבוא זה.

הידור הפרויקט לקובץ DLL

כאשר סיימת לבנות את פרויקט ActiveXDLL הכולל מחלקה (או מספר מחלקות עם או בלי טפסים ומודולים) ואשר מספק מאפיינים ושירותים שונים, עליך להדר את הפרויקט לקובץ dll. הידור הפרויקט לקובץ dll נעשה בעזרת האפשרות **Make Project1.dll** שבתפריט **File**. אפשרות זאת יוצרת קובץ dll המכיל את המחלקה שיצרת עם השירותים והמאפיינים אותה היא מספקת. בעזרת קובץ dll זה תוכל לשלב את יכולותיה של המחלקה בפרויקטים אחרים שתפתח או בכלי פיתוח שונים.

נניח לדוגמה כי אתה מפתח בכלי התומך באוטומציה אך אינו יכול לעבוד מול מסדי נתונים, מגבלה זו תוכל לפתור ביצירת קובץ dll. בקובץ DLL תבנה מחלקה אשר תספק את כל השירותים הנדרשים לעבודה מול מסד הנתונים. מתוך כלי הפיתוח תקרא לקובץ dll ובעזרת שירותיו תתקשר למסד הנתונים. במילים אחרות, את המגבלות של כלי הפיתוח (חוסר יכולת לעבוד מול מסד הנתונים) יפתור קובץ DLL שיצרת. אתה פשוט תקרא לשירותיו והוא יבצע עבורך את כל הפעולות להן מוגבל כלי הפיתוח בו אתה מפתח.

שים לב, קובץ פרויקט ActiveXDLL הוא פרויקט לכל דבר. משמעות הדבר שקובץ dll שתהדר ממנו לא יפעל אצל הלקוח לו תספק את הקובץ DLL. עליך לפתח לו ערכת התקנה (בדומה לקובץ EXE) אשר תתקין לו את הקובץ dll לעבודה בצורה מלאה (ללא תלות בסביבת הפיתוח של ויז'ואל בייסיק).

קובץ dll לדוגמה

כדי להטמיע את הנלמד עד כה, נבנה יחד קובץ dll פשוט, לדוגמה, המאפשר לך לקרוא מתוך קבצי טקסט. השירותים אותם יספק ה-dll הם:

- פתיחת קובץ (OpenFile). שירות זה יקבל כפרמטר את שם הקובץ ומסלולו אותו נרצה שיפתח ויחזיר כערך את מספרו המזהה של הקובץ.
- קריאת שורה מקובץ (ReadLine). שירות זה יקבל כפרמטר כתובת של משתנה מחרוזת אשר לתוכה יכתוב את השורה הנוכחית בקובץ, בו עומד המצביע, ואת מספרו הסידורי של הקובץ.
- סגירת הקובץ (CloseFile). שירות זה יקבל כפרמטר את מספרו המזהה של הקובץ ויסגור אותו.

בנוסף יספק dll מאפיין (EndFile) אשר יחזיר ערך True כאשר הגענו לסוף הקובץ וערך False במידה ולא.

לצורך בניית קובץ dll זה פתח פרויקט חדש מסוג ActiveXDLL.

המחלקה MyFile

כאשר נפתח הפרויקט (ActiveXDLL) הוא יוצר באופן אוטומטי מחלקה. שנה בחלון המאפיינים את המאפיין Name של המחלקה וקרא לה בשם MyFile. עתה הצהר על המשתנה blnEndOfFile מסוג Boolean :

```
Private blnEndOfFile As Boolean
```

שים לב, מכיון שזה משתנה, הצהרנו עליו כ-Private. מטרת המשתנה היא להודיע אם הגענו לסוף הקובץ או לא. אם הגענו לסוף הקובץ משתנה זה יקבל את הערך True.

מכיון שאנו משתמשים במשתנה במחלקה, ניצור מאפיין דרכו נוכל לקרוא את ערך המשתנה. בחר באפשרות **Add Procedure** שבתפריט **Tools** ובחר באפשרות **Property** והקלד את השם **EndFile** בתיבת הטקסט **Name** כדי ליצור מאפיין EndFile למחלקה.

שים לב, ה-dll בדוגמה שלנו אמור רק לספק מידע באשר לערך המשתנה blnEndOfFile ללא אפשרות עריכה שלו. במקרה זה מחק את החלק Let של המאפיין, והשאר רק את החלק Get בו נעשה שימוש.

מטרתו של המאפיין EndFile היא להודיע האם הגענו לסוף הקובץ או לא, או במילים אחרות מהו ערכו של המשתנה blnEndOfFile.

שורות הקוד של מאפיין זה תיראנה כך :

```
Public Property Get EndFile() As Boolean
   EndFile = blnEndOfFile
End Property
```

שירותי המחלקה

בשלב זה נבנה את שירותי המחלקה. כאמור, שלושת שירותי המחלקה הם : פתיחת קובץ, קריאת שורה מקובץ וסגירת קובץ.

פתיחת קובץ

כדי שקובץ dll שלנו יספק את האפשרות לפתיחת קובץ, נבנה את השירות **OpenFile**. שירות זה יקבל כערך את שם הקובץ ומסלולו אותו נרצה שיפתח, והוא יחזיר כערך את מספרו הסידורי של הקובץ שנפתח. מכיון ששירות זה מחזיר ערך ניצור אותו כפונקציה. אל תשכח ליצור את הפונקציה כ-Public כדי שזו תוכל להיקרא גם מחוץ ל-dll.

שורות הקוד של הפונקציה תיראנה כך:

```
Public Function OpenFile(ByVal strFileName As String)_
                                    As Integer
   Dim intFileNum As Integer
   intFileNum = FreeFile
   Open strFileName For Input As #intFileNum
   OpenFile = intFileNum
End Function
```

שים לב, אם קרתה שגיאה במהלך פתיחת הקובץ, שגיאה זו תופיע ביישום הלקוח של ה- dll או במילים אחרות, בפרויקט ממנו אתה עושה שימוש ב- dll.

קריאת שורה מקובץ

שירות נוסף אותו יספק ה-dll שלנו הוא: קריאת השורה הנוכחית בקובץ (השורה הנוכחית היא השורה עליה עומד מצביע הקלט בקובץ).

שירות זה (**ReadLine**) מקבל כפרמטרים מצביע למחרוזת, אליה הוא ישים כערך את השורה שנקראה מהקובץ, ואת מספרו המזהה של הקובץ. מכיון שהשירות אינו מחזיר ערך (כי הארגומנט שקיבל הוא כתובת המשתנה) נכתוב את השירות כשיגרה.

שורות הקוד של השירות ReadLine תיראנה כך:

```
Public Sub ReadLine(strLine As String, ByVal intFileNum _
                        As Integer)
   If blnEndOfFile = True Then Exit Sub
   Line Input #intFileNum, strLine
   If EOF(intFileNum) Then blnEndOfFile = True
End Sub
```

סגירת קובץ

השירות האחרון אותו יספק ה-dll שנבנה הוא **CloseFile**. שירות זה יקבל כפרמטר את מספרו הסידורי של הקובץ שנרצה שהשירות יסגור עבורנו.

שורות הקוד של השירות CloseFile הן:

```
Public Sub CloseFile(ByVal intFileNum As Integer)
   Close #intFileNum
   blnEndOfFile = False
End Sub
```

בזה סיימנו את כתיבת המחלקה וכתיבת הפרויקט כולו. לפני שנהדר את הפרויקט לקובץ dll, בחר באפשרות **Properties** שבתפריט **Project** ורשום בתיבת הטקסט Project Name את השם **MyDll** כשם הפרויקט. בתיאור הפרויקט (Project Description) רשום "**My Example DLL 1.0**". בחר באפשרות **Make MyDll.dll** שבתפריט **File** והדר את הפרויקט לקובץ dll.

השימוש ב-dll מכלי פיתוח אחר

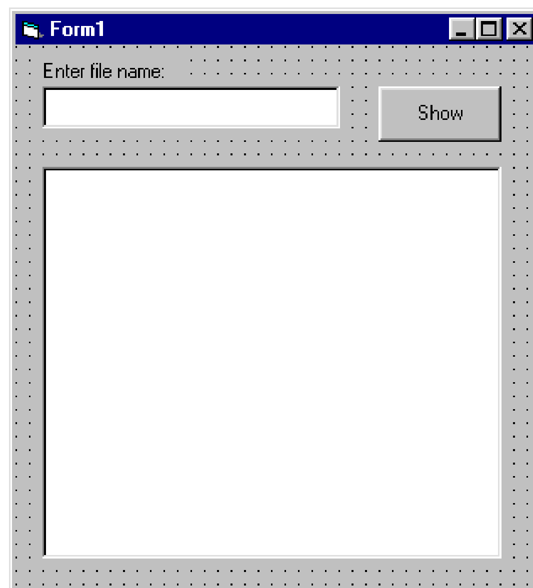
בשעה טובה יצרת קובץ dll המספק שירותים לכל כלי פיתוח התומך באוטומציה ובכלל זה ויז'ואל בייסיק. אתה בהחלט יכול לנצל את ה-dll שיצרת ולהשתמש בו בפרויקטים אחרים של ויז'ואל בייסיק שתיצור, או שיצרו חבריך, ואשר ייהנו מהשירותים שה- dll שלך יספק עבורם.

נניח, לצורך הדוגמה, כי ויז'ואל בייסיק שברשותך אינו מסוגל לקרוא מקובץ טקסט, ולצורך קריאה מקבצים אלו תיעזר ב- dll שכתבת.

פתח פרויקט סטנדרטי חדש וצור את הטופס המוצג בתרשים 13.3 וקבע את ערך מאפייני הפקדים בהתאם לטבלה הבאה:

טבלה 13.1

ערך	מאפיין	פקד
cmdShow Show	Name Caption	CommandButton
Enter file name:	Caption	Label1
txtFileName (Empty)	Name Text	Text1
txtFile (Empty) True 2 - Vertical	Name Text MultiLine ScrollBars	Text2



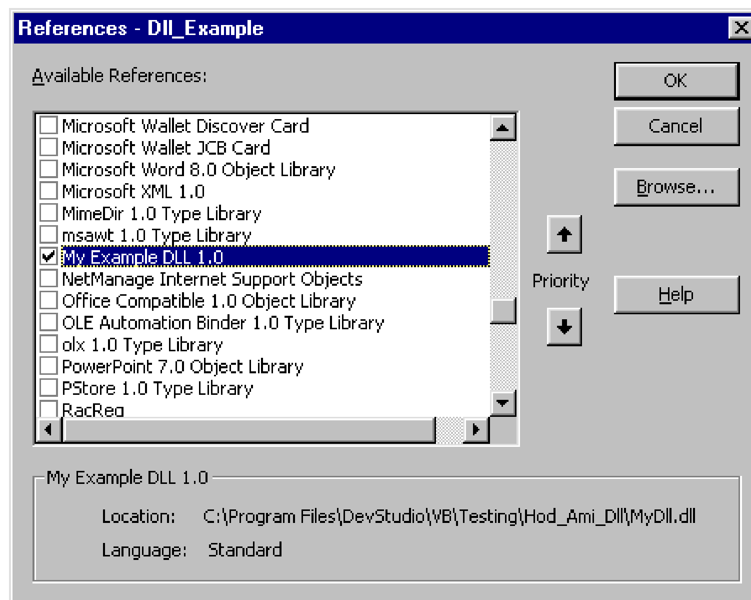
תרשים 13.3

בתיבת הטקסט: "Enter File Name:" הקלד את שם קובץ הטקסט ומסלולו. בלחיצה על הלחצן Show תוכן הקובץ יוצג בתיבת הטקסט התחתונה.

Project References

מכיון שהפרויקט (שוב לצורך הדוגמה) שאתה יוצר אינו מסוגל לקרוא מקבצים, היעזר בקובץ ה- dll שיצרת ובשירותיו.

כדי לאפשר לפרויקט שלך להיעזר בשירותיו של ה-dll (MyDll) שיצרת, בחר באפשרות **Project References** שבתפריט **Project** וסמן את תיבת הסימון שמשמאל ל- "My Example Dll 1.0" (תרשים 13.4). אם הוא לא מופיע היעזר בלחצן **Browse** כדי להוסיף את קובץ dll לתוך הרשימה. לאחר סימון התיבה לחץ על אישור כדי לאשר את הפעולה. פעולה זו הורתה לוויז'ואל בייסיק כי ברצונך להיעזר בשירותיו של ה- dll (MyDll) שיצרת.



תרשים 13.4

שימוש בשירותי ה- dll בקוד

בשלב זה עליך להצהיר על משתנה המייצג את המחלקה (שבקובץ dll) אשר במאפייניה ובשירותיה נעשה שימוש בפרויקט שלנו.

להצהרה על משתנה מסוג המחלקה MyFile שב- dll MyDll רשום (בחלק ה- General Declaration של הטופס):

```
Dim clsFile As MyDll.MyFile
```

שים לב , שסוג המשתנה MyDll.MyFile משלב בתוכו את שם הפרויקט שיצרנו (MyDll) ואת שם המחלקה שבו (MyFile).

כל שנותר הוא ליצור מופע של מחלקה זו ולקשר אותה עם המשתנה clsFile. יצירת מופע של אובייקט מחלקה זו, נעשית בעזרת הפונקציה CreateObject.

לדוגמה,

```
Private Sub Form_Load()
   Set clsFile = CreateObject("MyDll.MyFile")
End Sub
```

מעתה המשתנה clsFile מייצג את המחלקה (MyFile) שב-dll (MyDll), ודרכו נוכל להשתמש בשירותיו של ה- dll (או בעצם בשירותיה של המחלקה MyFile שבו).

בעזרת משתנה זה נשלים את שורות הקוד המתאימות. מטרת יישום הדוגמה שלנו היא, שכאשר נלחץ על הלחצן Show הוא יציג את תוכן הקובץ הרשום בתיבת הטקסט

"Enter File Name" בתיבת הטקסט התחתונה. עלינו, אם כן, לכתוב את שורות הקוד לאירוע Click של הפקד cmdShow.

שורות הקוד הן (שים לב, ויז'ואל בייסיק מזהה באופן אוטומטי את כל המאפיינים והשירותים אותם מספק ה- dll שיצרת):

```
Private Sub cmdShow_click()
   Dim intFileNum As Integer
   Dim strLine As String
   Dim strFileName As String
   StrFileName = txtFileName.Text
   intFileNum = clsFile.OpenFile(strFileName)
   Do While Not clsFile.EndFile
      Call clsFile.ReadLine(strLine, intFileNum)
      txtFile.Text = txtFile.Text & strLine & vbCrLf
   Loop
   clsFile.CloseFile intFileNum
End Sub
```

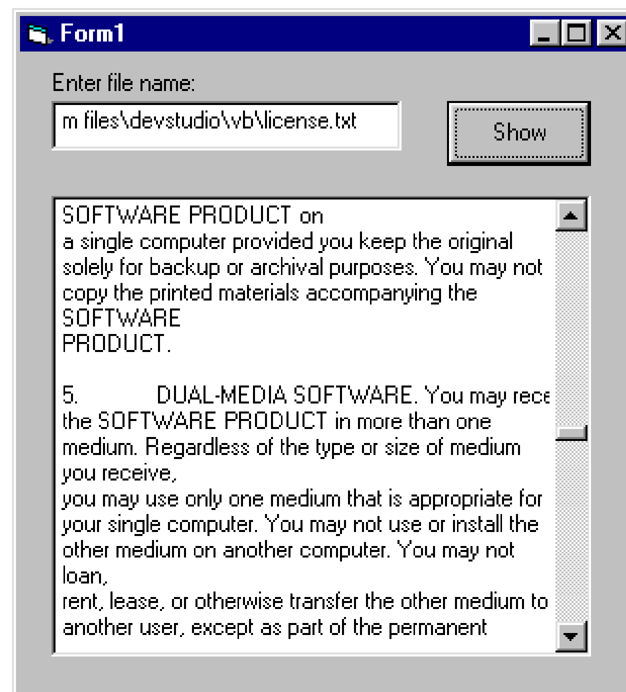
הסבר:

המשתנה clsFile מייצג את המחלקה MyFile שבקובץ ה-dll שיצרת. כל קריאה לשירות או מאפיין של המחלקה תיעשה דרך משתנה זה. כאשר המשתמש לוחץ על הלחצן Show יוצאות לפועל שורות הקוד שנכתבו לעיל לשיגרה cmdShow_Click. בשורות קוד אלו נעשה (אחרי הצהרת המשתנים) שימוש בשירות OpenFile של ה- dll (או של המחלקה MyFile שלו), כדי לפתוח את הקובץ אשר נרשם בתיבת הטקסט txtFileName. שים לב, היישום שלך לא יודע לפתוח קובץ והוא באמת לא זה שפתח את הקובץ. היישום שלך קרא לשירות OpenFile של ה- dll והוא פתח עבורו את הקובץ המתאים.

לאחר מכן יש לולאה הקוראת שורה אחר שורה מהקובץ עד לסופו. האינדיקציה לכך כי המצביע עומד בסוף הקובץ מתקבלת על ידי המאפיין EndFile (של ה- dll) המחזיר ערך True כאשר הגענו לסוף הקובץ. כל עוד הלולאה רצה (הקובץ לא הגיע אל סופו) מתבצעת קריאה של שורה מהקובץ. את קריאת השורה מהקובץ מבצע ה-dll על ידי קריאה לשירות ReadLine שלו.

בסיום הלולאה (כאשר נקרא כל תוכן הקובץ והועתק לתיבת הטקסט שביישום שלך) מתבצעת סגירה של הקובץ. שוב, גם את פעולה זו מבצע ה-dll (אל תשכח, יישום הדוגמה שלנו, אינו יודע לטפל בקבצים) בעזרת קריאה לשירות CloseFile שלו.

הקש F5 והרץ את היישום (תרשים 13.5).



תרשים 13.5

בחלק זה של הפרק למדת כיצד ליצור קובץ dll המספק יכולות רבות ומורכבות יותר לסביבות עבודה אשר כשלעצמן אינן יכולות לספק את השירותים הנדרשים. ניתן לנצל את קובץ ה- dll גם ליישום בוויז'ואל בייסיק, על אף שהוא יודע לבצע בעצמו את מה שה- dll עושה, כשכבה אמצעית לפרויקטים הבנויים בארכיטקטורת Tiers - 3.

באופן זה תוכל ליצור קבצי dll מורכבים יותר המספקים שירותים רבים יותר ולשלבם בסביבות פיתוח שונות התומכות באוטומציה כזו המאפשרת גישה ל- dll חיצוני.

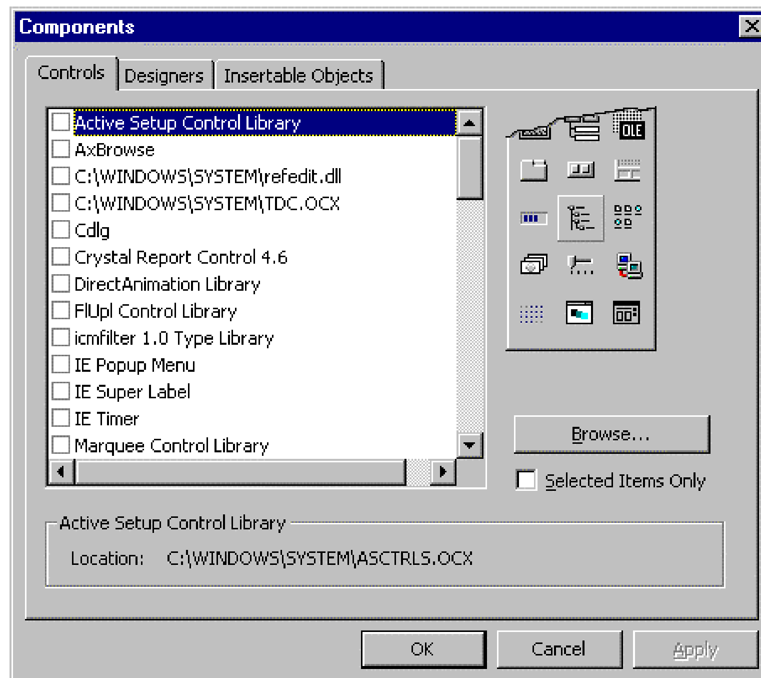
ActiveX Control

הפקדים שנמצאים בארגז הכלים של ויז'ואל בייסיק הם **מחלקות** (classes) המכילות מאפיינים ושירותים. כשאתה מציב בטופס פקד מסוג **תיבת טקסט** (TextBox), עשית שימוש במחלקה המייצגת את הפקד. כדי להעביר את המיקוד לפקד זה צריך להשתמש בשירות של המחלקה הנקרא **SetFocus**. הנה דוגמה:

```
Text1.SetFocus
```

מחלקות אלו מיוצגות על ידי **קובץ OCX**. קובץ זה מכיל מידע לגבי המחלקה, המאפיינים והשירותים שלה ומאפשר לשלבם בפרויקט ויז'ואל בייסיק.

לדוגמה, כדי להוסיף פקד לארגז הכלים, לוחצים עליו לחיצה ימנית ובוחרים באפשרות **Components**. בחלון מוצגת רשימת כל הפקדים הניתנים להוספה. השורה התחתונה (תרשים 13.6) מציגה את שם הקובץ המייצא את הפקד.



תרשים 13.6

רשימה זו כוללת את כל הפקדים הקיימים אצלך, אך אין אלה הקבצים היחידים. יש חברות רבות המפתחות ומספקות פקדים שונים, וכיום ישנם מאות פקדים המוצעים למכירה. לדוגמה, פקד המסייע לנו לשלוח E-Mail בקלות מתוך היישום בוויז׳ואל בייסיק ועוד. פקדים אלה הם קבצי OCX שתוכל להוסיף לארגז הכלים שלך.

בפרק זה נלמד כיצד לבנות פרויקט ActiveX Control בוויז׳ואל בייסיק ולהדרו לקובץ OCX. במילים אחרות, ויז׳ואל בייסיק מאפשרת לבנות פקד משלנו. בפקד נוכל להיעזר בכל פרויקט שנבנה. כל שעלינו לעשות, הוא להוסיף את הקובץ לרשימת Components שלנו ואת הפקד לארגז הכלים. בשלב זה ניתן להשתמש בו כפקד רגיל.

יצירת פרויקט ActiveX Control

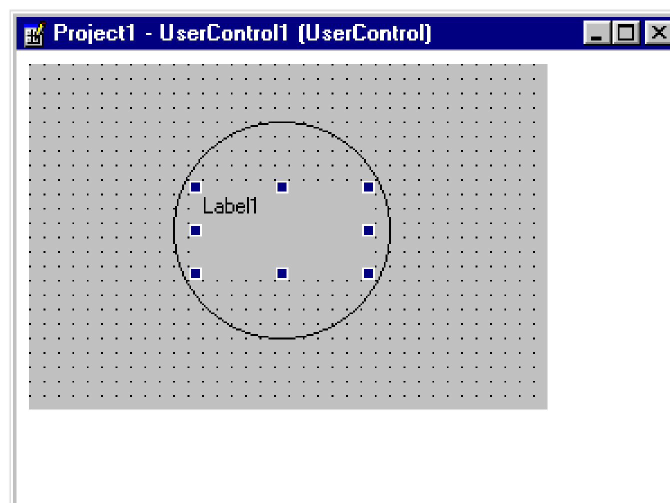
הפעל את ויז׳ואל בייסיק וצור פרויקט חדש. לפניך מופיע החלון **New Project**. עד כה התייחסנו לפרויקט מסוג Standard EXE, עתה בחר בפרויקט ActiveX Control.

לאחר שאישרת את הבחירה מופיע החלון **UserControl** הדומה במבנהו לטופס רגיל. ואכן UserControl זהה לחלוטין לטופס רגיל והעבודה בו, כהצבת פקדים, קביעת

מאפיינים וכתיבת הקוד זהה לעבודה מול טופס רגיל בוויז'ואל בייסיק. גם בעבודה עם **UserControl** יש לשמור על הסדר הרגיל, יצירת ממשק משתמש, קביעת מאפיינים וכתיבת הקוד.

בניית ממשק משתמש

בניית ממשק המשתמש ב- UserControl נעשה באופן דומה לממשק המשתמש שבנינת בטופס רגיל. בחר בפקד מתוך ארגז הכלים והצב אותו על גבי UserControl. כדוגמה, הנח על UserControl את הפקדים **Shape** ו- **Label** (תרשים 13.7).



תרשים 13.7

קביעת מאפיינים

קבע את מאפייני הפקדים על פי הטבלה הבאה:

פקד	מאפיין	ערך
UserControl	Name	ShapeLabel
Shape1	Name Shape	shpCirc Circle
	BackStyle	Opaque
Label1	Name Alignment BackStyle	lblCap Center Transparent

כתיבת הקוד

שטח **UserControl** הוא שטח הפקד שייווצר, אך כאשר מציבים אותו על הטופס הוא ניתן להגדלה, אם דרוש. כאשר לוחצים על הפקד במבט עיצוב נראות סביבו שמונה נקודות אחיזה המאפשרות להגדיל או להקטין אותו בכל כיוון. כאשר מגדילים את שטח הפקד בטופס, צורתו גדלה בהתאם לשטח שנקבע לו ע"י משיכת נקודות האחיזה.

אם נייחס את פעולות הגדלה והקטנה של שטח הפקד ל- UserControl, הגדלה או הקטנה של גודל הפקד הן שינוי גודל שטח UserControl כולו. כאשר משנים את גודל הפקד בטופס רגיל, הדבר מתבטא בכך שהפקדים המוצבים על UserControl משנים את גודלם בהתאם לגודל UserControl עצמו. בדוגמה שלנו, אם המשתמש יגדיל את הפקד (שטח UserControl), נצטרך לדאוג לכך שגם הפקדים Shape ו- Label יגדלו בהתאם.

פעולה זו נבצע בשורות קוד, אך מהו האירוע המתאים שבו נכתוב את שורות הקוד המתאימות? נצפה ששורות הקוד המשנות את גודל הפקדים Shape ו- Label, יבוצעו כאשר משתנה גודל השטח UserControl. אם כן, האירוע המתאים לכך הוא האירוע **Resize** של **UserControl**. בחר באירוע Resize (מתוך מבט הקוד) של UserControl וכתוב בשיגרה את שורות הקוד האלו:

```
Private Sub UserControl_Resize()
   shpCirc.Top = 0
   shpCirc.Left = 0
   shpCirc.Width = UserControl.Width
   shpCirc.Hight = UserControl.Hight
   lblCap.Top = UserControl.Hight / 2 - lblCap.Hight / 2
   lblCap.Left = UserControl.Width / 2 - lblCap.Width / 2
End Sub
```

בזה סיימנו את בניית הפרויקט, או את בניית הפקד. כל שנותר הוא לקבוע את מאפייני הפקד אשר יהיו נגישים למשתמש.

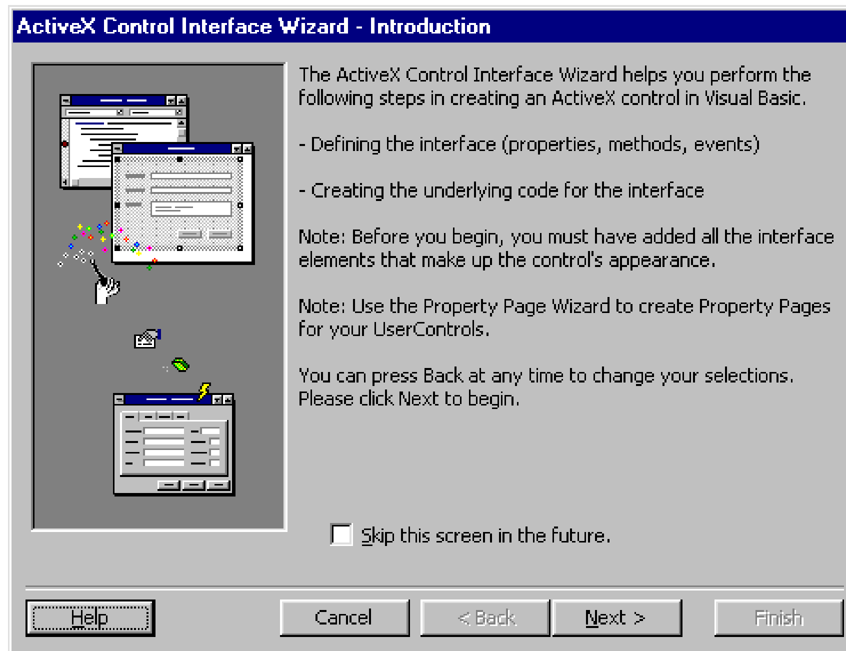
אשף ממשק הבקרה של ActiveX (ActiveX Control Interface Wizard)

בשלב זה צריך לקבוע איזה מאפיינים יהיו לפקד. למרות שזהו פקד הכולל את הפקדים Shape ו-Label, צריך להגדיר עבורו את המאפיין Caption (ל- Label) לדוגמה, כדי שהמשתמש **יוכל** לפנות אל המאפיין הזה בשעה שיציב את הפקד על הטופס.

לצורך קביעת המאפיינים ניעזר באשף **ActiveX Control Interface Wizard**, שבתפריט **Add-Ins**. אם אפשרות זו אינה קיימת, צריך להוסיף אותה. לצורך כך יש לבצע את הפעולות הבאות:

1. בחר באפשרות **Add-In Manager** שבתפריט **Add-Ins**.
2. מהחלון **Add-In Manager** בחר באפשרות **VB ActiveX Control Interface Wizard**.
3. בחר באפשרות **ActiveX Control Interface Wizard** שבתפריט **Add-Ins**.

בחירה באפשרות זו מפעילה את האשף (תרשים 13.8).



תרשים 13.8

קביעת מאפייני הפקד

לחץ על הלחצן **Next** ועבור לחלון **Select Interface Members** (תרשים 13.9).

בחלון זה יש שתי רשימות. הרשימה השמאלית **Available names** כוללת את כל המאפיינים והשירותים האפשריים לפקד זה; הרשימה הימנית **Selected names** כוללת את כל המאפיינים והשירותים הקיימים בפועל בפקד. ניתן להוסיף או לגרוע מרשימה זו בעזרת לחצני החיצים שבין שתי הרשימות.

לצורך הדוגמה שלנו, נוסיף לפקד את המאפיין Caption. מהתיבה השמאלית נבחר במאפיין זה ונוסיף אותו לרשימה הימנית.



תרשים 13.9

מיפוי המאפיינים

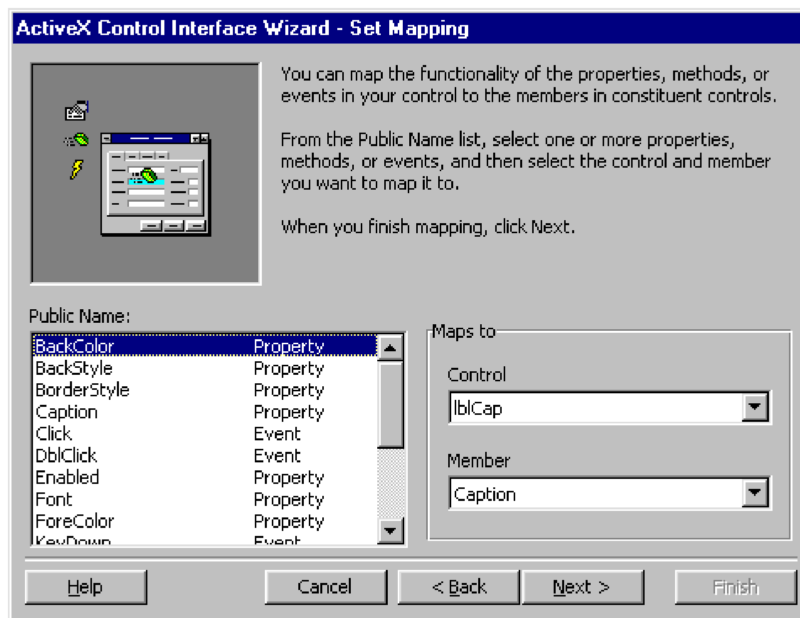
נלחץ פעמיים על **Next** ונעבור לחלון **Set Mapping** (תרשים 13.10).

בחלון זה נמפה את המאפיינים אל הפקדים שהוצבו ב- UserControl כדי לאפשר את השימוש בהם כאשר מציבים את הפקד על גבי הטופס.

במסגרת **Maps to** יש שתי תיבות משולבות. התיבה העליונה Control מתייחסת לפקדים (שכללת ב- UserControl), והשנייה - Member מתייחסת למאפיין המשויך לפקד.

נבחר מתוך הרשימה העליונה את הפקד **lblCap** ונצמיד לו את המאפיין **Caption**. לפני שמשייכים מאפיין לפקד צריך לבחור את המאפיין מתוך הרשימה השמאלית, Public name. באופן דומה נבחר בפקד **shpCirc** ובמאפיין **BackColor**.

לסיום דרושה לחיצה על הלחצן **Next** ולאחר מכן על **Finish**. בשלב זה יוצג חלון סיכום, לחיצה על **Close** תחזיר אותנו אל הפרויקט.



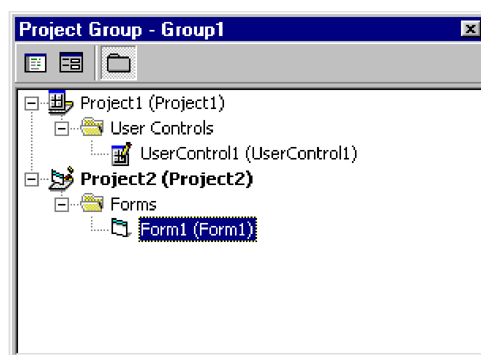
תרשים 13.10

בזה תמה מלאכת בניית הפקד וקביעת המאפיינים שלו. כל שנותר הוא לעשות בו שימוש בפרויקט רגיל של ויז'ואל בייסיק.

בדיקת הפקד

לאחר בניית פקד, יש לבדוק את תקינותו על ידי שילובו בפרויקט רגיל.

לחץ על הלחצן שבסרגל הכלים או בחר באפשרות **Add Project** מתפריט File. אפשרות זו תוסיף לפרויקט ActiveX Control של פרויקט סטנדרטי. חלון הפרויקטים שלך צריך להיראות כמודגם בתרשים 13.11:



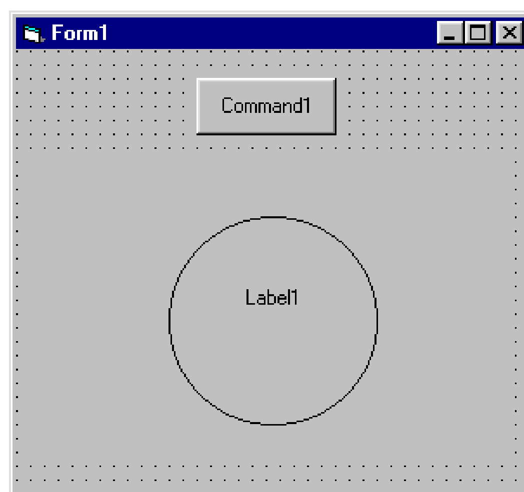
תרשים 13.11

מכיון שקיימים שני פרויקטים, ויז׳ואל בייסיק יוצרת **Group** באופן אוטומטי.

אם חלון UserControl שלך פתוח, סגור אותו ועבור לפרויקט הסטנדרטי. שים לב לתוספת בארגז הכלים. מלבד הפקדים הרגילים נמצא בו עתה פקד נוסף . פקד זה הוא UserControl, או במילים אחרות, הפקד שיצרת.

שילוב הפקד בפרויקט

ניצור עכשיו טופס הכולל את הפקד שיצרת ונבדוק אותו. לצורך כך נציב על הטופס (Form1) שני פקדים, את CommandButton ואת הפקד שיצרת (UserControl1) כמודגם בתרשים 13.12:



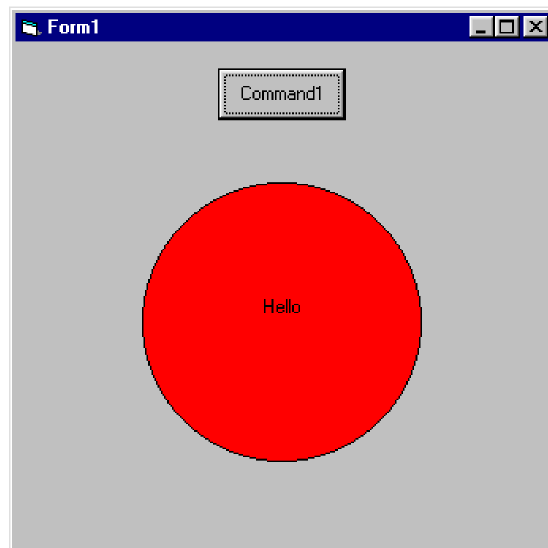
תרשים 13.12

בלחיצה על הלחצן Command1 נשנה את מאפייני הפקד החדש. נרשום את המילה ״Hello״ ככותרת (המאפיין Caption) ונצבע אותו באדום (BackColor = vbRed).

לצורך כך, נכתוב את שורות הקוד הבאות לאירוע Click של הפקד:

```
Private Sub Command1_Click()
    UserControl1.Caption = "Hello"
    UserControl1.BackColor = vbRed
End Sub
```

נפעיל את היישום (F5) ונלחץ על הלחצן (תרשים 13.13).



תרשים 13.13

הידור פרויקט ActiveX Control

אחרי שבדקנו ואימתנו את תקינות הפרויקט, יש להדרו וליצור קובץ OCX כדי שנוכל לשלב אותו כפקד בפרויקטים אחרים, או למסור אותו לאחרים.

כדי להקנות חזות מקצועית לפקד, נשנה את שמו ונקבע לו מאפיינים. לצורך כך נבחר באפשרות **Project Properties** מתפריט **Project** ונקבע את שם הפרויקט בדומה לפרויקט רגיל. לסיום נבחר באפשרות **Make Project1.ocx** מתפריט **File** כדי ליצור קובץ OCX.

בדומה ליצירת קובץ EXE גם בקובץ OCX יש לבנות ערכת התקנה, כדי שיהיה ניתן להשתמש בפקד גם במחשב שאין בו את סביבת הפיתוח של ויז׳ואל בייסיק. בניית ערכת ההתקנה נעשית באופן זהה לבניית פרויקט EXE סטנדרטי.

תרגול

1. בנה קובץ DLL (פרויקט ActiveXDLL) אשר ינהל מסד נתונים. קובץ ה- DLL יספק שלושה שירותים: פתיחת מסד נתונים, שליפת מידע מטבלה או טבלאות וסגירת מסד הנתונים.
2. בנה פקד (קובץ OCX) בעזרת פרויקט ActiveX Control. הפקד יהיה ComboBox אשר יציג את הנתונים שבו בסדר ממוין. הפקד יאפשר חיפוש מהיר. לדוגמה, הקשת אות אחת תציג מייד את המילה הראשונה שמתחילה באות זו וכן הלאה.

14 : אובייקטים ויישומי Microsoft Office

ויז'ואל בייסיק מאפשרת לקשר את היישום שאנו בונים אל יישומי Microsoft Office, כדוגמת Word ו- Excel. יש מספר דרכים להתקשר ליישומים חיצוניים מתוך ויז'ואל בייסיק, בעזרת OLE, בעזרת אובייקטים ועוד. בפרק זה נלמד את המבוא לעבודה עם אובייקטים, מכיון שהם נותנים כלים טובים לניהול יישומי Microsoft Office מתוך ויז'ואל בייסיק.

ויז'ואל בייסיק מספקת סוג משתנים בשם **Object** (אובייקט, עצם) ובעזרתו ניתן להפעיל יישומים שונים מתוך תוכנית ויז'ואל בייסיק. הדרך בה נעשה הקישור הוא על ידי הגדרת היישום שאנו רוצים לנהל כאובייקט בעזרת השירותים והפונקציות שהוא מספק. ראוי לציין שלא ניתן להתקשר לכל יישום בדרך זו. יישומי **Microsoft Office** תומכים בדרך קישור המוצגת בפרק זה, וניתן לנהל אותם כאובייקטים מתוך ויז'ואל בייסיק.

הצהרה על משתנה אובייקט

ככל משתנה רגיל יש להצהיר על משתנה מסוג **Object** לפני השימוש בו. מלבד אופן ההצהרה הזהה, משתנה זה מתנהג כמשתנה לכל דבר בכל הקשור לאורך חיים וטווח הכרה (ראה הסבר בפרק העוסק במשתנים).

ההצהרה על אובייקט:

```
Dim objMyDocument As Object
```

בשורת קוד זו הצהרנו על המשתנה objMyDocument כעל אובייקט שעדיין לא נעשה קישור שלו ליישום כלשהו.

הפונקציה CreateObject

כדי לקשר בפועל את האובייקט ליישום, ניעזר בפונקציה **CreateObject** שתפקידה ליצור **ActiveX Object**, אובייקט ActiveX. הפרמטר שמקבלת הפונקציה הוא שם המחלקה המייצגת את סוג האובייקט שדרכו נעשה הקישור. כל יישום התומך בסוג קישור כזה מספק לפחות סוג אחד של אובייקט שנוכל ליצור מוויז'ואל בייסיק.

השורות הבאות מדגימה יצירת אובייקטים שונים:

```
Set objMyDocument = CreateObject("word.basic")
Set objMyDocument = CreateObject("Excel.Application")
```

לאחר שנוצר האובייקט, המשתנה objMyDocument מייצג את היישום שאליו הוא מקושר. כל פנייה אל היישום תיעשה מעתה דרך המשתנה objMyDocument.

ניהול מסמך Word מוויז׳ואל בייסיק

אחד השימושים הנפוצים באובייקטים הוא ניהול מסמכי מעבד התמלילים Microsoft Word מתוך יישום וויז׳ואל בייסיק. לדוגמה נניח שפיתחת יישום המנהל מסד נתונים באמצעות תוכנה של חברה כלשהי. אחת האפשרויות שמספק היישום היא הצגת רשימת כל הלקוחות החייבים כסף לחברה. על פי רשימה זו המזכירה שולחת מכתבים לכל הלקוחות ומציינת את הפרטים האישיים ואת פירוט חובם לחברה. כתיבת מכתב בעל נוסח אחיד לכל הלקוחות, אשר שונים רק בפרטים האישיים ובגובה החוב, יכולה להיות עבודה מייגעת. כאן נוכל לנצל את היכולת של ויז׳ואל בייסיק לנהל מסמכי Word ולהדפיס את אותו מכתב מספר פעמים, ובכל פעם לשתול בו את נתוני הכתובת המתאימים הנלקחים ממסד הנתונים. זהו יישום פשוט של מיזוג דואר.

ניהול המסמך ייעשה מתוך שורות הקוד בוויז׳ואל בייסיק, אך תחביר הפונקציות והשימוש בהם לשילוב עם האובייקט אינו שפת ויז׳ואל בייסיק טהורה. בדוגמה שלנו התחביר בו נשתמש בשורות הקוד הוא של **VBA**, או Visual Basic for Applications.

טיפ: תחביר שפת VBA אינו חלק מוויז׳ואל בייסיק ואף אינו מובא במערכת העזרה של התוכנה, אם כי במקומות בודדים ובאופן כללי בלבד. כדי לדעת מהו התחביר בו תוכל להשתמש, הקלט מאקרו במעבד התמלילים Microsoft Word ואחר כך עיין בקוד שלו במבט עריכה. התחביר הכתוב שם הוא התחביר בו תוכל להשתמש (עם שינויים קלים) בשורות הקוד שלך בוויז׳ואל בייסיק.

השירות EditFind

נחזור לדוגמה ונראה מהן שורות הקוד המאפשרות לשתול נתונים מתוך מסד הנתונים למסמך Word, ובמקרה זה - למזג דואר.

השיטה המקובלת היא ליצור מסמך שבו תהיינה מילים שישמשו תבניות שבהן "יישתלו" הנתונים ממסד הנתונים. לדוגמה, נניח שתרצה לשתול את שם הלקוח ליד המילה "עבור:". במקרה זה תיצור מסמך שבו יהיה כתוב "עבור: <שם>". התוספת "<שם>" לאחר המילה "עבור:" היא התבנית שתוחלף בשם האמיתי של הלקוח שנלקח ממסד הנתונים. כל שנותר להורות ליישום הוא לחפש במסמך את התבנית "<שם>" ולשתול במקומה את השם הרצוי.

שורות הקוד הבאות מדגימות את שתילת המידע בתוך המסמך:

```
Dim MyWord As Object
Dim db As Database
Dim recCustomers As Recordset
Dim strSQL As String
Set db = OpenDatabase("c:\MyWork.mdb")
strSQL = "Select * from Customers Where Balance > 0"
Set MyWord = CreateObject("word.basic")
Set recCustomers = db.OpenRecordset(strSQL)

MyWord.InsertFile Name:="MyDocument.doc", Range:="", _
ConfirmConversions:=0, Link:=0

MyWord.EditFind Find:="<שם>", Direction:=0, MatchKashida:=0, _
MatchDiacritics:=0, MatchAlefHamza:=0, MatchControl:=0, _
MatchCase:=0, WholeWord:=0, PatternMatch:=0, SoundsLike:=0, _
Format:=0, Wrap:=1, FindAllWordForms:=0

MyWord.Insert recCustomers!LastName & "  " & _
   recCustomers!FirstName

MyWord.appshow
```

באופן דומה ניתן לשתול מידע בטבלאות, להדפיס מסמך למדפסת ועוד. לצורך זה יש להכיר את תחביר שפת VBA, אשר חורגת ממסגרת פרק זה.

הדוגמה שמוצגת בפרק זה מיועדת לעבודה במעבד התמלילים Word 7. בגירסה 97 יש שינויים קטנים אותם יש לבצע באופן כללי בהשוואה לגירסה 7. אופן העבודה זהה גם ל-Excel.

אם אינך מכיר מספיק טוב את שפת VBA, הנה טיפ קטן: כדי לדעת מהם שורות הקוד אשר יבצעו פעולות מסוימות במעבד התמלילים, פתח אותו והקלט מאקרו חדש. בתוך המאקרו בצע את אותן פעולות אשר היישום שלך אמור לבצע ושמור את המאקרו. עתה הצג את המאקרו במצב עריכה. במצב זה יוצגו לפניך שורות הקוד בשפת VBA אשר יוציאו לפועל את אותן פעולות שביצעת במאקרו. העתק שורות אלו לשורות הקוד שלך בוויז'ואל בייסיק. השינויים שתידרש לבצע יהיו מעטים מאוד. באופן זה תוכל לשלב בשורות הקוד שלך שורות קוד בשפת VBA, על אף שאינך בקיא מספיק בתחביר השפה.

תרגול

1. צור את הטופס הבא:

Form1

תרגיל ביצירת אובייקטים.
צור את הטופס הזה.
לחיצה על הלחצן Show תגרום לפתיחת מעבד
התמלילים Microsoft Word. את כל הכתוב בתיבת
טקסט זו תשתול התוכנית אל תוך המסמך
שנוצר.

Show

לחיצה על הלחצן Show תגרום לפתיחת מעבד התמלילים Microsoft Word. את כל הכתוב בתיבת הטקסט תשתול התוכנית אל תוך המסמך שנוצר.

2. בצע את תרגיל 1 אך הפעם המסמך לא יופיע, תוכנו יודפס ישירות למדפסת.

15 : מבוא לשפת SQL

שפת **SQL** ‏(Structure Query Language) - **שפה לניהול מסד נתונים טבלאי** - היא שפה תקנית ומקובלת כיום בכל מערכות התוכנה והמחשבים. היא מאפשרת הקמה וניהול של מסד נתונים, עדכון, הפקת שאילתות, הדפסה, יבוא, יצוא ועוד. השימוש בשפת SQL נפוץ מאוד לניהול מסדי נתונים, בין השאר גם במערכת Microsoft Access, ולפיכך נכיר אותה לפני שנעבור ללימוד הפרק הבא העוסק במסדי נתונים.

שפת SQL הפכה במרוצת השנים לשפה סטנדרטית. למרות שיש הבדלי תחביר בין הגרסאות של יצרנים שונים, הכלים הבסיסיים דומים בכל גרסאות השפה.

סוגי שאילתות

השם SQL קשור ליכולת הטובה של השפה לאחזר נתונים ממסדי הנתונים על פי שאילתות המוצגות על ידי המשתמש. השאילתות מחולקות לשני סוגים עיקריים: שאילתות בחירה ושאילתות ביצוע.

ב**שאילתות בחירה** (**Select**) מטרת השאילתה לקבל מידע ממסד הנתונים. לדוגמה, להפיק את רשימת שמות העובדים שהתקבלו לעבודה בשנתיים האחרונות. **שאילתת ביצוע** מבצעת שינוי במסד הנתונים, כמו עדכון, מחיקה ועוד. לדוגמה, לעדכן בתוספת של 10% את מחירי כל המוצרים שמחירם לא עודכן בשנה האחרונה.

שאילתת בחירה (Select)

שאילתת בחירה (Select) מיועדת לשליפת מידע ממסד הנתונים. המידע המתקבל משאילתת Select מציג את הנתונים הקיימים במסד הנתונים בלבד, ואינו מבצע בהם שינויים. לדוגמה, הצג את השם הפרטי ואת שם המשפחה של כל העובדים הקיימים בטבלת העובדים. המידע המתקבל מציג את הנתונים בלבד, ולא נעשה שינוי באחד משדות אלה במסד הנתונים.

תחביר השאילתה:

```
Select  field1 ,field2 , ..., fieldN
From  table
```

לדוגמה:

```
Select First_Name, Last_Name From Employees
```

ניתן לכתוב את השאילתה בשורה אחת, אך בדרך כלל כותבים את משפט Select בשורות אחדות שבראש כל אחת מהן נמצאת מילת מפתח, כפי שדרוש: From, Where, Order by ועוד.

דוגמה נוספת:

```
Select CustomerId, CompanyName
From Customers
```

רשימת השדות במשפט Select הם אלה שהמשתמש רוצה לראות אותם בתצוגת המסך או בהדפסה. בחלק From נרשם שם הטבלה שממנה נלקחים שדות אלה. וכאשר רוצים להציג שדות בסדר רצוי כלשהו, צריך לרשום את שמותיהם בסדר הרצוי. כדי לבחור את השדות שבטבלה, אפשר לרשום "*" (כוכבית) בלבד. לדוגמה, שאילתה להצגת כל השדות שבטבלה Employees:

```
Select * From Employees
```

קריטריונים (Where)

שאילתת Select פשוטה מציגה את הנתונים שבכל הרשומות בטבלה המבוקשת. אך מה קורה אם נרצה מידע מרשומות מסוימות בלבד העונות על קריטריון מסוים? לדוגמה, נכתוב שאילתה שמציגה מתוך רשימת העובדים רק את אלה שגילם 25 ומעלה. במקרה זה נשתמש במשפט **Where**. חלק Where במשפט SQL מייצג את הקריטריון, ולכן הרשומות שיתקבלו כתשובה מהשאילתה יתייחסו רק לרשומות שעונות על קריטריון זה בלבד.

תחביר המשפט הוא:

```
Select  field1, field2, ..., fieldN
From Table
Where Criteria
```

לדוגמה:

```
Select First_Name, Last_Name, Age
From Employees
Where Age >35
```

בדוגמה זו השאילתה תחזיר כתשובה את השם הפרטי ואת שם המשפחה של כל העובדים מטבלת Employees שגילם מעל 35.

אופרטורים להשוואה

בחלק Where במשפט SQL ניתן להשתמש לבניית הקריטריון באופרטורים להשוואה שונים. אופרטורים להשוואה הם:

1. > (גדול)
2. >= (גדול או שווה)

לדוגמה:

```
Select First_Name, Last_Name, Age
From Employees
Where Age >= 35
```

3. < (קטן)
4. <= (קטן או שווה)
5. <> (שונה)

לדוגמה:

```
Select First_Name, Last_Name, City
From  Employees
Where  City  <> 'Tel-Aviv'
```

בדוגמה זו מתקבלות כפלט כל רשומות העובדים ש**אינם** גרים בת״א.

אופרטורים לוגיים

באופרטורים להשוואה משתמשים כאשר הקריטריון הוא קריטריון פשוט. כדי ליצור קריטריון מורכב, ניעזר באופרטורים לוגיים:

1. And (וגם)

לדוגמה:

```
Select First_Name, Last_Name, Age
From Employees
Where Age > 35 And City = 'Jerusalem'
```

בדוגמה זו מתקבלות כל הרשומות של העובדים מטבלת Employees, אשר גרים בירושלים **וגם** גילם מעל 35. בדוגמה זו צירפנו בתנאי אחד שני קריטריונים פשוטים שבהם השתמשנו באופרטורי השוואה.

2. Or (או).

לדוגמה:

```
Select * From Employees Where Age > 35 or City = 'Tel-Aviv'
```

בדוגמה זו מתקבלות כל הרשומות של העובדים מטבלת Employees, שגרים בתל-אביב **או** אלה אשר גילם מעל 35, או שניהם יחד. בדוגמה זו, תושב ירושלים בן 42 ייכלל ברשימה.

3. not (הפוך מ..)

לדוגמה:

```
Select * From Employees Where Not(city = 'Tel-Aviv')
```

שאילתה זו מקבילה לשאילתה הבא:

```
Select * From Employees Where city <> 'Tel-Aviv'
```

קריטריונים נוספים

שימוש נוסף בקריטריונים הוא חיפוש ערך בתוך טווח ערכים.

1. Between....And לחיפוש ערך בטווח המוגדר. לדוגמה:

```
Select * From Employees Where Id Between 1 And 100
```

שאילתה זו מציגה את כל הרשומות של העובדים מטבלת Employees, אשר מספר ת״ז שלהם הוא בין 1 ל- 100 (ועד בכלל).

2. In לחיפוש ערך בתוך רשימת ערכים. לדוגמה:

```
Select * From Employees
Where Id In(1,7,13,90,102)
```

אם מספר הזהות הוא אחד מהערכים הנמצאים ברשימת ה-In הרשומה תוצג, אחרת היא לא תתקבל כתשובה לשאילתה.

יש פונקציות רבות אשר ניתן להשתמש בהם לבניית קריטריון השאילתה (חלק Where של משפט Select) כדי למקד את הקריטריון לצרכינו. תמיד רצוי לבחון היטב את תחביר מסד הנתונים המסוים שממנו מאחזרים את הנתונים.

סוגי משתנים

חלק Where במשפט Select, הקובע את הקריטריון, אפשר לכלול את כל סוגי השדות שבטבלה: מספרי, מחרוזת ותאריך. עם זאת, השימוש בהם בקריטריון שונה.

- כאשר הקריטריון כולל שדות אלפאנומרים, הערך להשוואה יושם בין שני גרשִים (' '). לדוגמה:

```
Select * From Employees
Where  City = 'Jerusalem'
```

ניתן להחליף את המחרוזת במשתנה (String) המכיל את המחרוזת בקריטריון, כמו בדוגמה זו. לדוגמה:

```
Dim strCity As String
strCity  = 'Jerusalem'
Select * From Employees Where City = strCity
```

- כאשר הקריטריון כולל שדות ותאריך, הערך יושם בין שתי סולמיות (#). לדוגמה:

```
Select * From Employees
Where Birth = #01/05/75#
```

גם כאן ניתן להחליף את הערך במשתנה מסוג Date.

- כאשר השימוש בקריטריון כולל מספרים, אפשר לכתוב את הערך המספרי ללא תיחום. לדוגמה:

```
Select * From Employees
Where Age >35
```

שאילתות סיכום

שאילתה מסוג נוסף היא שאילתת **סיכום**, שתפקידה לסכם ערכים מרשומות שונות. עושים זאת על ידי פירוט השדות, או על ידי פונקציית סיכום. לדוגמה, מהו מספר כל העובדים בחברה (על פי הטבלה Employees), או מהו סכום המשכורות הכולל שמשולם לעובדים אלה.

שאילתת סיכום כוללת שני חלקים:

א. פונקציית הסיכום (סכום, מניה וכו').

ב. קיבוץ השדות לקבוצות שעליהן יופעל הסיכום.

לדוגמה, מהו סכום הסטודנטים בכל הפקולטה. סכום הסטודנטים הוא פונקציית הסיכום והפקולטה היא שדה קיבוצי. הנה דוגמה מעשית לשאילתת סיכום:

```
Select Count(Id)
From Employees
GroupBy DepId
```

בדוגמה זו הרשומות מקובצות לפי השדה DepId (המייצג מספר מחלקה). במילים אחרות המידע מקובץ לפי מחלקות, כאשר המידע המבוקש הוא count(Id). בהפעלת הפונקציה count אנו מבקשים בעצם מהשאילתה שתספור את המספר הרשומות שבכל חלק קיבוצי. במילים אחרות, התשובה שתתקבל מהשאילתה תהיה - מהו מספר העובדים שבכל מחלקה ומחלקה.

כדי לדעת מהו סכום המשכורת (של כל העובדים) בכל מחלקה נרשום את משפט SQL הבא:

```
Select Sum(Salary)
From Employees
GroupBy DepId
```

שים לב, גם בסוג זה של שאילתה ניתן להגדיר קריטריון לרשומות שיתקבלו כתשובה. לדוגמה:

```
Select Sum(Salary), city
From Employees
Where city = 'Tel-Aviv'
GroupBy DepId
```

שאילתות על מספר טבלאות

עד כה עסקנו בשאילתות המספקות מידע מטבלה אחת. במשפט שאילתה אחד ניתן לשלוף מידע ממספר טבלאות. השוני בין מבנה השאילתה על מספר טבלאות לבין מבנה השאילתה על טבלה אחת הוא בכך שצריך לציין את שם הטבלה שהשדה שייך לה וגם צריך לציין את שם השדה המקשר בין שתי הטבלאות.

תחביר השאילתה הפועלת על כמה טבלאות:

```
Select table1.field1, table1.field2, ..., tableN.fieldM
From  table1, table2, ..., tableN
Where table1.field1 = table2.field1
```

לדוגמה:

```
Select Employees.First_Name, Departments.Dep_name,
   Departments.Manager, Departments.DepId
From Employees, Departments
Where Employees.DepId = Departments.DepId
```

בדוגמה זו מציגים כפלט את שם העובד, את שם המחלקה, את שם מנהל המחלקה ואת מספר המחלקה. הקריטריון לקישור בין שתי הטבלאות הוא מספר המחלקה המופיע כשדה הן בטבלת העובדים והן בטבלת המחלקות. גם בשאילתה זו ניתן להוסיף קריטריון ברירה של הרשומות באמצעות הרחבה של משפט Where. לדוגמה:

```
Select Employees.First_Name, Employees.Id,
   Department.Manager
From Employees, Departments
Where Employees.DepId = Department.DepId
   And Department.DepId =7
```

בדוגמה זו, נוסף לקריטריון התנאי שהפלט יכלול את העובדים במחלקה 7 בלבד.

שאילתות ביצוע

שאילתות ביצועיות שונות משאילתות בחירה (Select), כי הן מבצעות פעולות במסד הנתונים. לדוגמה, עדכון משכורת של עובדי הניקיון (קוד 4) בעיריית ירושלים, על ידי הוספת בונוס בתקופת מצעדים.

תחביר השאילתה הוא:

```
Update table
Set field = value
```

לדוגמה, תוספת 10% למשכורת:

```
Update Employees
Set Salary = Salary * 1.1
Where city = 'Jerusalem' and work = 4
```

שים לב, אם לא היה קריטריון - העדכון היה תקף לכל הרשומות בטבלה. לדוגמה:

```
Update Employees
Set  Salary = Salary * 1.1
```

במקרה זה משכורתם של כל עובדי המפעל היתה מתעדכנת בתוספת של 10%.

שאילתת מחיקה

שאילתה זו משמשת למחיקת רשומות מטבלה.

תחביר השאילתה הוא:

```
Delete
From table
```

לדוגמה:

```
Delete
From Employees
```

שים לב, במקרה זה יימחקו **כל** הרשומות שבטבלה, מכיון שלא נקבע קריטריון מיוחד. אם לדוגמה, רוצים לפטר עובד מסוים (בעל ת"ז 123) נציין זאת במפורש בקריטריון כדי שהשאילתה תמחק רק את הרשומה העונה על הקריטריון הזה בלבד. לדוגמה:

```
Delete
From Employees
Where Id = 123
```

תרגול

לפניך שתי טבלאות במסד נתונים כלשהו :

Departments
Id
Description

Employees
Id
Fname
Lname
BirthDate
Address
Tel
DepId

1. כתוב שאילתת SQL אשר תציג את כל פרטי העובדים.
2. כתוב שאילתת SQL אשר תציג את שמם המלא ואת מספר הטלפון של העובדים במחלקת מכירות.
3. כתוב שאילתת SQL אשר תציג את שמם המלא של כל עובדי מחלקת ההנהלה, אשר יום הולדתם חל החודש.

16 : מסדי נתונים

מסד נתונים (DataBase) הוא אוסף של סוגי נתונים שונים הקשורים ביניהם בקשרים לוגיים כלשהם, ומאפשר את שיתוף הנתונים בין יישומים שונים. במאגר הנתונים הזה נשמרים נתונים שקודם לכן נהגו לשמור בקבצים נושאיים. לשמירת הנתונים בקבצים נושאיים, כלומר קובץ לכל נושא ותפקיד, היו חסרונות רבים שגרמו לבניית מערכת חלופית לשמירת הנתונים – מסד הנתונים.

באופן כללי ניתן לדמות את מבנה מסד הנתונים למבנה של תיקיית ספרים. מסד הנתונים מזוהה על ידי שמו, שהוא למעשה "שם קובץ" במושגים של מערכת ההפעלה. בתוך התיקיה נמצאים ארונות המכילים ספרים בנושאים שונים: יש ארון "ספרי מחשב", ארון "ספרי פסיכולוגיה", ארון "ספרי מדע בדיוני" וכו'. עם זאת, הכל נמצא במסגרת של תיקיה אחת, וכאן – מסד נתונים אחד.

הנתונים שנשמרים במסד הנתונים מחולקים ל"**טבלאות**" (tables) שבכל אחת מהן יש נתונים הקשורים לנושא שאותו היא מייצגת. לדוגמה, במסד נתונים המכיל נתונים מידע על מפעל מסוים, יימצאו טבלאות עובדים, המכילים את כל המידע הקשור לעובדים, טבלת מוצרים, המכילים את כל המידע הקשור למוצרים וכו'.

נחזור לתיקיה, כל ארון מחולק למדפים, המייצגים תת-נושאים בתוך הנושא הראשי הכולל את כל הספרים בארון. למשל, בארון מחשבים, נמצא ספרי מחשבים העוסקים בלימוד שפות תכנות, ספרים אחרים עוסקים בתקשורת מחשבים ועוד. באופן דומה, כל טבלה במסד הנתונים בנויה מ"**רשומות**" (records) רבות, שכל אחת מהן מכילה את כל המידע לגבי פריט אחד בטבלה. לדוגמה, בטבלת עובדים, פרטי העובד ששמו דני הם רשומה אחת, פרטיו של משה הם רשומה שנייה, פרטיה של שרה הם רשומה שלישית וכן הלאה.

בתחתית ההיררכיה נמצא את הספר (שבגללו נכנסנו בכלל לתיקיה). אם נרצה לשאול ספר בנושא ויז'ואל בייסיק נפנה לארון ספרי מחשבים, למדף המכיל ספרים העוסקים בלימוד שפות תכנות ושם נמצא בין הספרים השונים את הספר העוסק בוויז'ואל בייסיק. מקביל לו במסד הנתונים הוא ה"**שדה**" (field). כל רשומה מורכבת מכמה שדות (לפחות אחד) וכל השדות יחדיו מהווים את כל המידע המרכיב את הרשומה. למשל, בטבלת עובדים יש רשומה האוגרת את פרטי העובד משה. בשדה אחד נשמור את שמו הפרטי, בשדה שני את שם המשפחה, שדה נוסף לת"ז וכך הלאה (תרשים 16.1).

עובדים : טבלה

שם פרטי	שם משפחה	תז	כתובת	טלפון
משה	כהן	012345670	הרצל 11 ירושלים	02-6543210
דוד	לוי	076543212	אלנבי 33 ת"א	03-9876541
ישראל	ישראלי	115987532	האומן 8 ירושלים	02-3698741

רשומה: 3 מתוך 3

תרשים 16.1

הכלי המאפשר לנו להציג את הנתונים האגורים במסד הנתונים, להוסיף נתונים, לערוך, לעדכן ולמחוק ממנו נתונים, הוא **הפקד Data**. פקד זה פועל יחד עם "מנוע" מסד הנתונים Microsoft Jet (זהו המנוע של מסד הנתונים שמפעיל את התוכנה לניהול מסדי נתונים Microsoft Access). בעזרתו יש לנו גישה למסד הנתונים ויכולת לבצע בו שינויים במידת הצורך.

הפקד Data מורכב למעשה משני פקדים. הראשון הוא **Data Access Control** המאפשר גישה למסדי נתונים, כגון: Microsoft Access, Btrieve, dBASE, Microsoft FoxPro ו-Paradox.

כברירת מחדל, הפקד מתקשר אל מסד נתונים של Access, ואם נרצה לקשר את הפקד למסד נתונים אחר (כדוגמת Paradox, FoxPro, dBASE ועוד) נציין זאת במפורש במאפיין Connect.

הפקד השני הוא **Remote Data Control** המאפשר גישה למסד נתונים מרוחק כמו למשל, Microsoft SQL Server ו-Oracle.

הפקד Data

הפקד Data מאפשר גישה למסד הנתונים כדי להוסיף, לערוך, למחוק ולעדכן ללא צורך בכתיבת שורות קוד כלשהן. יש שני מאפיינים בסיסיים שהמשתמש נדרש לקבוע את ערכם, כדי שהפקד יבצע את פעולתו. שני המאפיינים הם: DatabaseName ו-RecordSource.

המאפיין **DatabaseName** קובע את שם מסד הנתונים שהפקד מקושר אליו, או שם קובץ ונתיב של מסד הנתונים. ברגע שנבחר מסד הנתונים ונקבע במאפיין DatabaseName, מתקבלת רשימת ערכים אוטומטית במאפיין **RecordSource**, והיא מכילה את רשימת כל הטבלאות שנמצאות באותו מסד נתונים. מתוך רשימה זו נבחרת טבלה אחת המהווה את ערך המאפיין RecordSource.

שים לב שפקד Data יכול להתקשר אך ורק לטבלה אחת במסד הנתונים, אולם מספר פקדי Data שניתן להציב בטופס אחד אינו מוגבל.

המאפיין **RecordSource** מכיל את הטבלה שהפקד מציג את פרטיה. הוא יכול לעשות זאת בשלוש דרכים המיוצגות על ידי שלושת ערכי המאפיין **RecordsetType**.

Table-type-Recordset. ערך זה קובע כי הפקד יציג טבלה אחת, זו שרשומה במאפיין RecordSource. ניתן להוסיף, לשנות, לערוך ולמחוק את הנתונים שבטבלה.

Dynaset-Type-Recordset הוא ערך ברירת המחדל. ערך זה מאפשר לפקד להציג טבלה יחידה מתוך מסד הנתונים או להציג תוצאת שאילתה (המכילה מספר שדות מטבלה אחת או יותר). גם כאן ניתן להוסיף, לשנות, לערוך ולמחוק נתונים והשינויים יחולו על הטבלה או הטבלאות המתאימות.

SnapShot-type-Recordset. ערך זה מציין כי הפקד מציג מעין צילום של שדות מטבלה אחת או יותר. מה שניתן הוא רק לצפות בנתונים בלבד. אין אפשרות לערוך נתונים אלה.

ניתן לקשר פקדים נוספים לפקד Data כדוגמת תיבת טקסט, תיבת רשימה, רשת ועוד, אשר יציגו את נתוני מסד הנתונים שהפקד Data מקושר אליהם. בפרק זה לא נעסוק בכך ורק נציין שכדי לקשר פקד כתיבת טקסט אל פקד Data יש לקבוע ערכים למאפיין DataSource הקובע לאיזה פקד Data מוצמדת תיבת הטקסט; ולמאפיין DataField הקובע איזה שדה בטבלה יוצגו ערכיו בתיבת הטקסט.

אולם יכולת הפקד Data להתקשר למסד נתונים לצורך קריאה, כתיבה ועדכון מסד נתונים מוגבלת, בעיקר מבחינת ממשק המשתמש. ביישומים מורכבים הפועלים עם מסדי נתונים דרושה מעורבות באמצעות שורות קוד בתוכנית. על כן, יוחד פרק זה לעבודה מול מסד נתונים באמצעות שורות הקוד.

בניית מסד נתונים באמצעות ויז׳ואל בייסיק

כדי לבנות מסד נתונים אנו זקוקים לתוכנה המאפשרת זאת. לדוגמה, מסד נתונים MDB (זו סיומת ״קובץ״ מסד הנתונים) נבנה בעזרת התוכנה Microsoft Access. שים לב שלצורך הגישה למסד הנתונים משורות הקוד בוויז׳ואל בייסיק אין צורך בתוכנה זו. שורות הקוד פונות ישירות אל מסד נתונים קיים. התוכנה Access דרושה **ליצירת** מסד הנתונים בלבד. אם אין לך כלי מיוחד לבניית מסד נתונים, ויז׳ואל בייסיק מספקת כלי כזה; זוהי התוכנית **VisData**.

התוכנית VisData

תוכנית VisData היא יישום ליצירת מסד נתונים. התוכנית מאפשרת ליצור מסד נתונים מסוגים שונים כגון: Microsoft Access, Dbase, FoxPro ועוד.

התוכנית VisData היא יישום עצמאי וניתן להפעילה באופן עצמאי ולא דרך ויז׳ואל בייסיק. קובץ התוכנית **VisData.exe** נמצא בתיקיה שבה מותקנת ויז׳ואל בייסיק. לחילופין ניתן להפעיל את התוכנית מתוך ויז׳ואל בייסיק. כדי להפעיל את התוכנית VisData מתוך ויז׳ואל בייסיק, בחר באפשרות **Visual Date Manager** שבתפריט **Add-Ins**. בחירה זו פותחת את הקובץ VisData.exe ומציגה לפנינו את מסך הפתיחה של התוכנית VisData (תרשים 16.2).



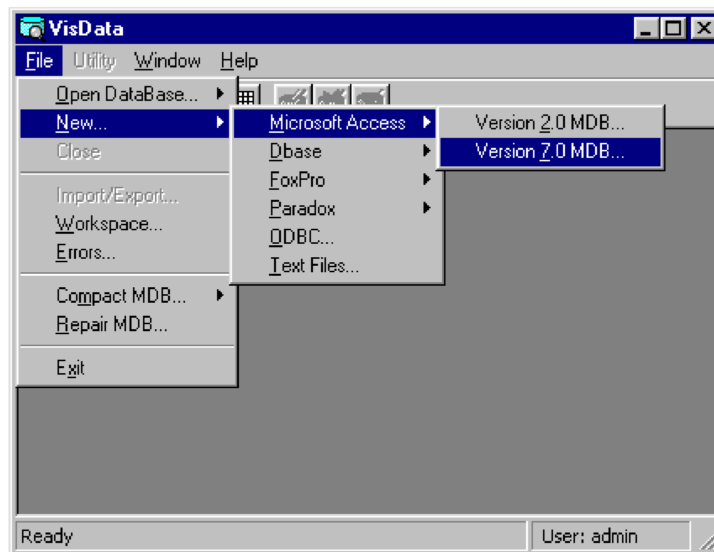
תרשים 16.2

יצירת מסד נתונים

בשלב הראשון יש ליצור מסד נתונים חדש. כדי ליצור אותו בצע את הפעולות הבאות:

1. בחר באפשרות **New** שבתפריט **File**. בחירה זו מציגה מספר סוגים של מסדי נתונים.
2. בחר בסוג מסד נתונים הרצוי ובגירסה הרצויה (אם קיימות מספר גרסאות). כדוגמה, בחר במסד הנתונים Microsoft Access (תרשים 16.3). בחירת סוג מסד נתונים גורמת לפתיחת חלון בו תתבקש לתת שם למסד הנתונים, שזה למעשה שם הקובץ שנוצר.

3. הקש בתיבת הטקסט את שם הקובץ מסד הנתונים ואשר את הבחירה. יצירת הקובץ גורמה ליצירת מסד הנתונים ולהצגת שני חלונות: Database Window - לצורך בניית מסד הנתונים ו-SQL statement לצורך שאילתות.



תרשים 16.3

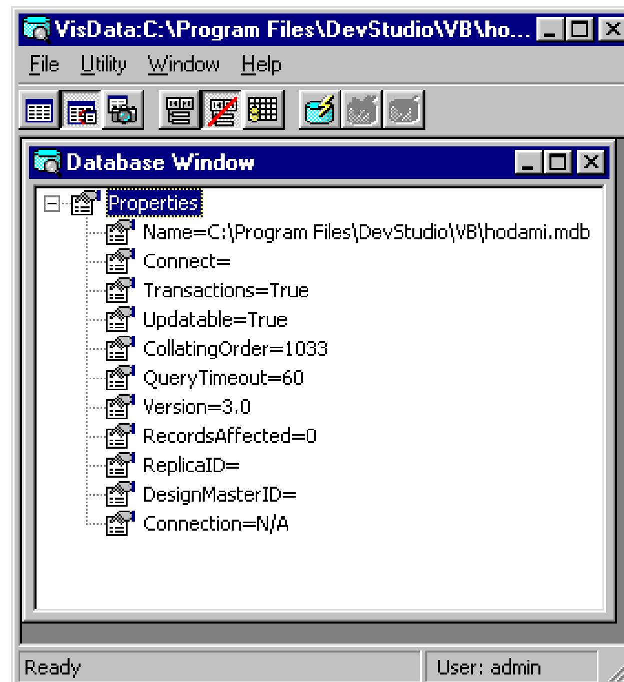
בניית טבלה

כדי לבנות את מסד הנתונים ניעזר בחלון Database Window (תרשים 16.4).



תרשים 16.4

חלון זה מציג עץ בעל ענף אחד בשם **Properties**, אשר מכיל את מאפייני מסד הנתונים שכרגע נוצר. חלון זה דומה לחלון הסייר של Windows: לחיצה על + ("פלוס") שנמצא משמאל לענף גורמת לפתיחת הענף ולהצגת מאפייני הנתונים (תרשים 16.5). לחיצה נוספת באותו מקום (שסימנו כעת הוא -) סוגרת את הענף.



תרשים 16.5

בשלב הראשון יש לבנות את הטבלאות המרכיבות את מסד הנתונים. כדוגמה, נבנה שתי טבלאות: **Employees** המכילה נתונים על עובדים במפעל (שם פרטי, שם משפחה, ת"ז וקוד מחלקה). הטבלה השנייה היא **Departments** המכילה נתונים על המחלקות במפעל (קוד מחלקה, שם מחלקה, מנהל מחלקה ומספר עובדים).

לבניית הטבלה Employees בצע את הפעולות הבאות:

1. לחץ לחיצה ימנית על הענף **Properties** שבחלון **Database Window**.
2. בחר באפשרות **New Table**. בחירה זו גורמת לפתיחת החלון **Table Structure** (תרשים 16.6).
3. בתיבת הטקסט **Table Name** הקלד את שם הטבלה **Employees**.

בשלב זה יצרנו את הטבלה ונתנו לה שם. בשלבים הבאים נוסיף בה את השדות הדרושים.

Table Structure

Table Name:

Field List: | Name: | Type: | FixedLength | Size: | VariableLength | CollatingOrder: | AutoIncrement | AllowZeroLength | OrdinalPosition: | Required | ValidationText: | ValidationRule: | DefaultValue:

Add Field | Remove Field

Index List: | Name: | Primary | Unique | Foreign | Required | IgnoreNull | Fields:

Add Index | Remove Index

Build the Table | Close

תרשים 16.6

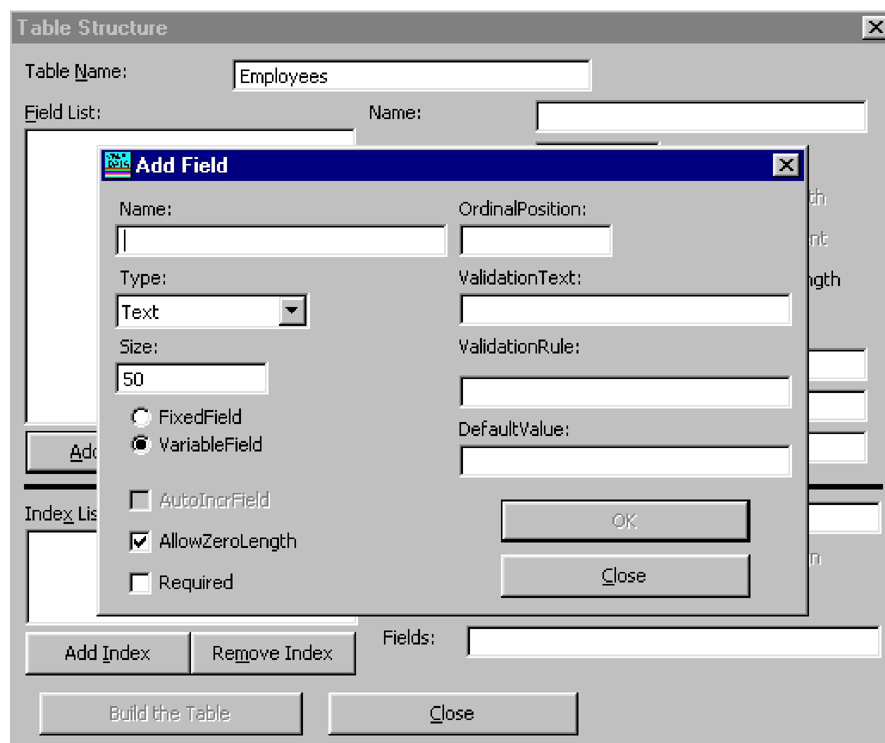
הוספת שדות לטבלה

להוספת השדה שם פרטי בצע את הפעולות הבאות:

1. לחץ על הלחצן **Add Field**.

   לחיצה זו גורמת להצגת החלון Add Field (תרשים 16.7) בחלון זה נמלא נתונים על השדה.

2. בתיבת הטקסט **Name** רשום את שם השדה **FirstName**.
3. בתיבה הנפתחת **Type** בחר את סוג השדה. בדוגמה שלנו הוא **Text**.



תרשים 16.7

4. בתיבת הטקסט **Size** הקלד את גודל השדה. גודל 20 לדוגמה, קובע כי השדה FirstName יכיל עד 20 תווים.

תיבות הסימון AllowZeroLength ו-Required קובעות האם השדה יכול להכיל מחרוזת ריקה (כלומר להיות ריק) והאם חובה למלא בו ערך, בהתאמה.

אם נרצה לקבוע קריטריון לערכים המוקלדים בשדה, נקבע זאת בתיבת הטקסט **ValidationRule**. לדוגמה, שדה מספרי מסוג Integer מכיל ערכים מספריים. נניח שנרצה להגביל את הערכים לאלה הגדולים מ- 10, במקרה זה, בתיבת הטקסט ValidationRule נרשום את הביטוי "‎>10" (גדול מ- 10). מעתה כל ערך שיוקלד בשדה ולא יתאים לדרישה "גדול מ- 10" יגרור אחריו הודעת שגיאה. את הודעת השגיאה קובעים בתיבת הטקסט **ValidationText**. מה שנרשם בתיבה זו יוצג אם הערך שהוכנס לשדה עובר על "חוק האימות".

בתיבת הטקסט **DefaultValue** הקלד ערך אשר יהווה ערך ברירת המחדל. שים לב, אין חובה למלא תיבה זו, אלא אם נרצה שיופיע ערך ברירת המחדל בתוך השדה.

5. לסיום אשר את נתוני השדה בלחיצה על הלחצן **OK**.

באותו אופן נוסיף לטבלה את השדות LastName, Id ו-DepCode ובסיום נלחץ על **Close** כדי לחזור לחלון **Table Structure** (תרשים 16.8). כל שדות הטבלה מוצגים בחלון Table Structure. כדי למחוק אחד מהשדות, נבחר בו מתיבת הרשימה **Field List** ונלחץ על הלחצן **Remove Field**.

תרשים 16.8

אינדקסים

תפקיד האינדקסים במסד הנתונים לאפשר אחזור מהיר של נתונים. חסרון האינדקס בכך שהוא מאט כתיבה של רשומות חדשות במסד הנתונים. כששדה בטבלה משמש כאינדקס, משמעות הדבר שהשדה הזה מוחזק כטבלה ממוינת. כאשר המשתמש מזין נתונים למסד הנתונים הם נשמרים בו בסדר כרונולוגי של הזנתם. במילים אחרות, הם נמצאים בצורה לא ממוינת, אלא אם המשתמש הכניס את הנתונים בצורה ממוינת. חיפוש נתון שתואם לקריטריון מסוים יהיה איטי, מכיון שהרשומות אינן ממוינות. אם לדוגמה נרצה לחפש מספר ת״ז של עובד מתוך כל הרשומות בטבלת עובדים, המחשב יצטרך לעבור רשומה רשומה ולבדוק אם היא תואמת לקריטריון. אם לא התמזל מזלנו, הרשומה התואמת היא הרשומה האחרונה, וכך יתנהל לו החיפוש לאורך כל הרשומות הקיימות.

לעומת זאת, אם שדה ת״ז היה מוגדר כשדה אינדקס, מסד הנתונים היה בונה לו מעין טבלה בזיכרון (שאינה נראית למשתמש או למתכנת) ושומר שם את הערכים בסדר ממוין. בכל פעם שבטבלה המקורית מתוספת רשומה בעלת ערך חדש של ת״ז, רק מספר ת״ז זה נשמר אוטומטית באופן ממוין בטבלת האינדקס. בנוסף לשדה ת״ז שומרת התוכנית לניהול מסד הנתונים בזיכרון גם מצביע אל הרשומה המקורית, כדי שאפשר יהיה לדעת היכן היא נמצאת.

מכיון שבעת הכנסת הנתונים צריך לטפל גם בטבלת האינדקס הממוינת, זמן הכתיבה של נתונים חדשים הוא איטי. לעומת זאת, כאשר נרצה לאחזר נתונים, כמו למשל שם של עובד מסוים, ננצל למטרה זו את האינדקס ונפנה אל הרשומה על פי שדה ת״ז. במקרה זה החיפוש לא נעשה באופן כרונולוגי, מכיון ששדה ת״ז הוא שדה אינדקס והחיפוש נעשה מטבלת האינדקס שנמצאת בזיכרון וממוינת. בדרך זו החיפוש מהיר יותר ושליפת הנתונים מהירה יותר.

שדה אינדקס מיוחד נקרא **מפתח** (key) שפועל בדיוק כמו אינדקס רגיל. המיוחד בו שאין הוא מרשה כפילות ערכים. לדוגמה, אם שדה ת״ז היה מפתח, התוכנה לא תרשה ששתי רשומות בטבלת העובדים תהיינה בעלות מספרי ת״ז זהים. שדה מפתח הוא שדה שערכו הוא **ייחודי** (unique). שם פרטי לדוגמה, לא יכול להיות שדה מפתח מכיון ששני עובדים ויותר יכולים להיות בעלי אותו שם פרטי.

לאינדקס מפתח יש שם מיוחד, המייחד אותו משאר האינדקסים: **PrimaryKey** (מפתח ראשי). שים לב שבטבלה אחת ייתכנו כמה אינדקסים, אך יכול להיות בה מפתח אחד בלבד. שדה מפתח יכול להיות מורכב מכמה שדות וכך גם שדה האינדקס.

הוספת אינדקסים

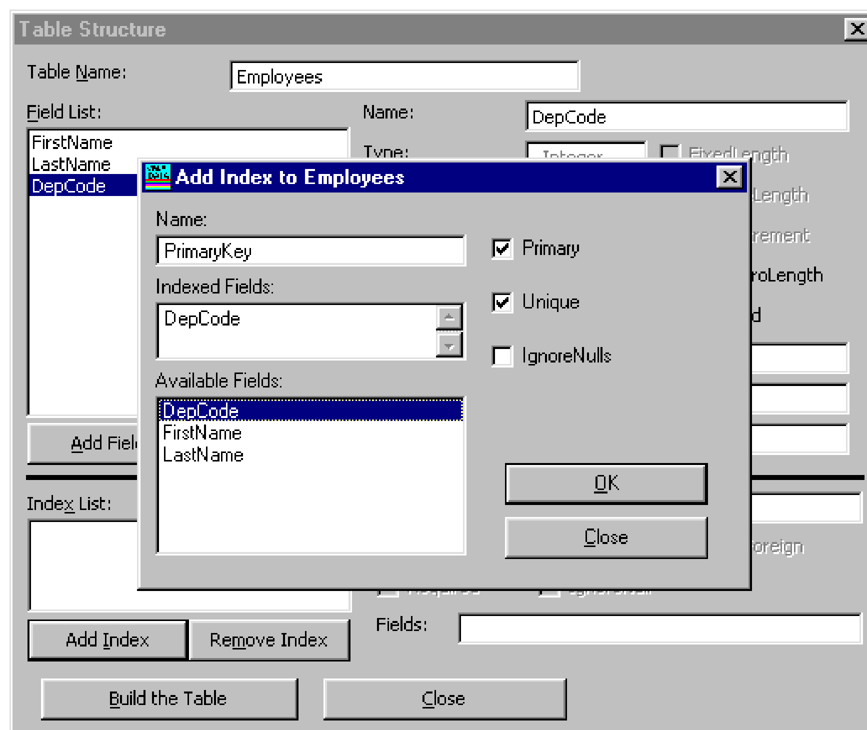
נחזור לחלון **Table Structure**. חלון מאפשר להוסיף אינדקסים לטבלה. כדי להוסיף אינדקס לטבלה בצע את הפעולות הבאות:

1. לחץ על הלחצן **Add Index**.

   בחירה זו גורמת לפתיחת החלון **Add Index To Employee**.

2. בתיבת הטקסט **Name** הקלד שם לאינדקס.
3. מתוך תיבת הרשימה **Available Fields** בחר את תיבת הסימון **Primary**. שים לב שבטבלה אחת יכול להיות רק אינדקס (מפתח) ״PrimaryKey״ אחד בלבד.
4. אשר את הנתונים בלחיצה על לחצן OK. חזור על הפעולות על כל אינדקס נוסף שתיצור (תרשים 16.9).
5. לחץ על הלחצן **Close** כדי לסיים את יצירת האינדקסים ולחזור לחלון Table Structure.

התיבה Index List מכילה עכשיו את שמות האינדקסים שנוצרו.



תרשים 16.9

לבניית הטבלה לחץ על לחצן **Build The Table**. פעולה זו סוגרת את החלון ומחזירה אותנו לחלון Database Window, לאחר שענף התוסף תחת הענף Properties ונושא את שם הטבלה החדשה.

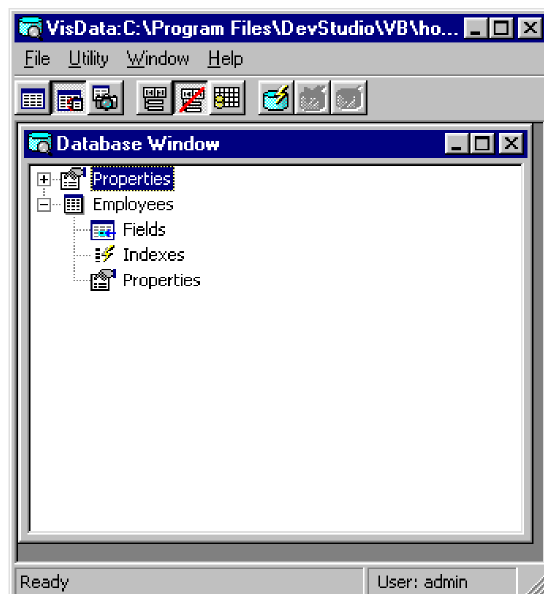
עריכת טבלה

מחלון **Database Window** ניתן לצפות בטבלה ובמרכיביה (השדות, האינדקסים, והמאפיינים שלה). לחיצה על סימן + משמאל לענף הטבלה גורם להרחבת הענף ולהצגת שלושה תת-ענפים:

Fields - להצגת שדות.

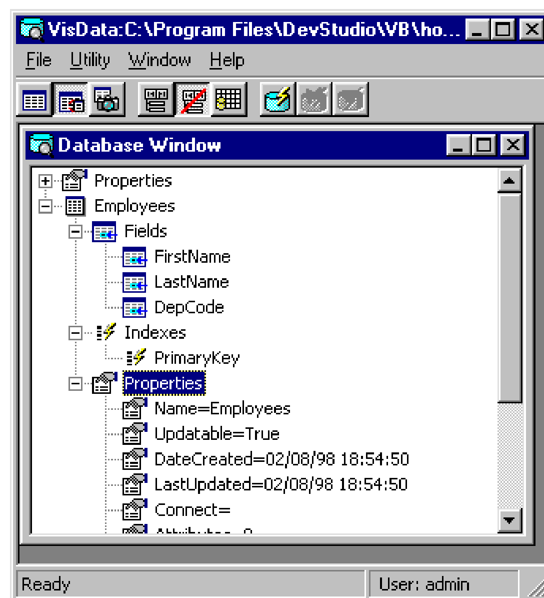
Indexes - להצגת האינדקסים.

Properties - להצגת מאפייני הטבלה (תרשים 16.10).



תרשים 16.10

כל תת ענף יכול להתרחב אף הוא לתת-ענפים המרכיבים אותו, וכך הלאה. לדוגמה, הענף Fields מציג תחתיו את רשימת השדות הקיימים בטבלה, וכל שדה מציג תחתיו את רשימת המאפיינים של השדה הנבחר (תרשים 16.11).



תרשים 16.11

ניתן לעשות שינויים בטבלה ולערוך אותה מחדש, ניתן למחוק את הטבלה, לשנות את שמה, לפתוח אותה במטרה למלא בה נתונים ועוד. לביצוע כל זה, דרושה לחיצה ימנית על ענף הטבלה להצגת תפריט מקוצר שבו אפשרויות שכיחות. ארבע האפשרויות הראשונות הן:

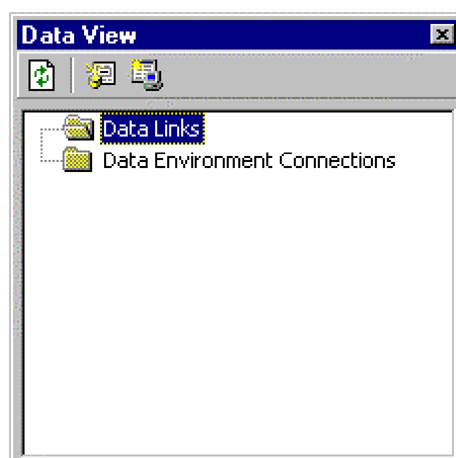
1. Open - לפתיחת הטבלה למטרת הזנת נתונים.
2. Design - פתיחת הטבלה במבט עיצוב (החלון Table Structure) לעריכת הטבלה מחדש.
3. Rename - לשינוי שם הטבלה.
4. Delete - למחיקת הטבלה.

באופן דומה נבנה את כל הטבלאות והשדות במסד הנתונים. בסיום העבודה נבחר באפשרות **Exit** שבתפריט **File** כדי לצאת מהתוכנה VisData ולחזור לוויז'ואל בייסיק.

בוויז'ואל בייסיק 6 נוספה אפשרות בה ניתן לנהל מסד נתונים עם טבלאות, קשרים, שאילתות באופן גרפי קל וברור. השיפור המשמעותי בגירסה 6 הוא היכולת לנהל כמה מסדי נתונים שונים, כדוגמת Microsoft SQL Server ו-Microsoft Access למשל.



הלחצן (Data View Window) שבסרגל הכלים מציג את ממשק המשתמש, בו נוכל לנהל את מסד הנתונים החדש, כמוצג בתרשים הבא:



תרשים 16.12

ניהול מסד הנתונים משורות הקוד

כדי לנהל מסד נתונים מתוך תוכנית וויז׳ואל בייסיק, צריך להיעזר במנוע התוכנה **Microsoft Jet Database**. לתפעול המנוע מספקת ויז׳ואל בייסיק את האובייקט **DAO** (Data Access Object) שאליו נתייחס בפרק זה. האובייקט **RDO** (Remote Data Object) שמאפשר לנהל מסד נתונים מרוחק באמצעות **ODBC** למשל, אך באובייקט זה לא נדון במסגרת ספר זה. אובייקט נוסף לניהול מסד נתונים הוא **ADO** (ActiveX Data Object). אובייקט זה דומה במידה רבה לאובייקט DAO ובתפעולו – לפקד Data, וניתן לבנות פקד כזה באופן ידני בעזרת ADO. נראה שמטרת האובייקט ADO להחליף את האובייקט DAO.

קישור היישום עם מנוע מסד הנתונים

האובייקט DAO מכיל אוסף של פונקציות, מאפיינים ושירותים, ובכלל זה סוגי משתנים לעבודה מול מסד נתונים. כדי לנהל מסד נתונים באמצעות האובייקט DAO יש להגדיר זאת במפורש כחלק ממאפייני הפרויקט. לצורך כך בחר באפשרות **Reference** **שבתפריט** **Project** וסמן את תיבת הסימון שמשמאל ל- Microsoft DAO 3.51 Object Library.

הצהרה על משתנים

כמשתנים רגילים, גם משתנים המשתמשים לניהול מסד נתונים מוצהרים קודם השימוש בהם. יש סוגים שונים של משתנים המיועדים לניהול מסד הנתונים. בפרק זה נתמקד בשכיחים שבהם, בסוג המשתנה **Database** לניהול מסד נתונים ובסוג המשתנה **Recordset** לניהול טבלה או אוסף רשומות. ההצהרה על משתנה כזה דומה להצהרה על משתנה רגיל:

```
Dim db As Database
Dim rst As Recordset
```

פתיחת מסד נתונים

לאחר ההצהרה על המשתנים נפתח בעזרתם את מסד הנתונים הרצוי. פתיחת מסד נתונים נעשית בעזרת השירות **OpenDatabase**, לפי התחביר הבא:

```
Set db = OpenDatabase("C:\Vb\Nwind.mdb")
```

במשפט זה נשתמש לפתיחת מסד נתונים Nwind.mdb במשתנה db מסוג Database. מעתה, המשתנה db מייצג את מסד הנתונים Nwind.mdb וכל התייחסות למשתנה db היא למעשה התייחסות למסד הנתונים Nwind.mdb. כפי שניתן לראות, אפשר להשתמש במשתנה String כפרמטר לשירות OpenDatabase.

לדוגמה:

```
Dim strDbName As String
strDbName = "C:\Vb\Nwind.mdb"
Set db = OpenDatabase(strDbName)
```

הפונקציה ()CurDir

כאשר פותחים מסד נתונים (או קובץ רגיל), שולחים לפונקציה או לשירות ארגומנט, שהוא שם הקובץ והנתיב. נניח שבנינו את היישום בתיקיה MyProg שבתיקיית השורש, ובה נמצא מסד הנתונים שאנו פונים אליו. לדוגמה:

```
Dim db As Database
Set db = OpenDatabase("C:\MyProg\MyDb.mdb")
```

הכל טוב ויפה עד שאנו מוסרים את היישום שלנו בתוספת ערכת התקנה ללקוח והוא מחליט שברצונו להתקין את היישום (וממילא את קובץ מסד הנתונים MyDb.mdb) בתיקיה אחרת, C:\HisProg. במקרה זה נתקלים בבעיה, מכיון שכאשר התוכנית מבצעת את המשפט הבא:

```
Set db = OpenDatabase("c:\MyProg\MyDb.mdb")
```

היא אינה מוצאת את הקובץ MyDb.mdb כי הוא אינו נמצא בתיקיה MyProg אלא בתיקיה HisProg. התוצאה: כישלון התוכנית, עם הודעת שגיאה שהקובץ לא נמצא!

כדי למנוע תופעה לא נעימה זו, אין להתייחס בקוד לתיקיה מוגדרת, אלא לתיקיה כללית. ברור לנו כי קובץ מסד הנתונים MyDb.mdb נמצא בתיקיה שבה נמצא היישום שלנו. אם כן, בכל פעם שנרצה לפנות לקובץ מסד הנתונים נאמר לתוכנית שתחפש אותו בתיקיה הנוכחית ובמילים אחרות, בתיקיה בה פועל הקובץ. בכל מקרה, קובץ היישום וקובץ מסד הנתונים נמצאים באותה תיקיה, ובתיקיה זו נחפש את הקובץ.

כדי להורות לתוכנית לחפש את הקובץ בתיקיה הנוכחית או בתיקיית היישום, קיימות בוויז'ואל בייסיק שתי דרכים. האחת היא שימוש בפונקציה **( )CurDir** המחזירה כערך את שם התיקיה (נתיב שלם) הנוכחית, בה נמצאת מערכת ההפעלה. לשם תיקיה זו נוסיף את שם קובץ מסד הנתונים MyDb.mdb ונבקש מהתוכנית לפתוח אותו. לא חשוב באיזו תיקיה התוכנית מותקנת – הקובץ יימצא. לדוגמה:

```
Dim db As Database
Dim strFileName As String
strFileName = CurDir() & "\MyDb.mdb"
Set db = OpenDatabase(strFileName)
```

דרך נוספת היא השימוש במאפיין **Path** של האובייקט **App**. האובייקט App מכיל נתונים שונים על היישום שלנו. המאפיין Path מאחסן כערך את שם התיקיה (נתיב שלם) של היישום. חיפוש הקובץ MyDb.mdb ייעשה כעת בתיקיית היישום הנכונה. לדוגמה:

```
Dim db As database
Dim strFileName As String
strFileName = App.Path & "\MyDb.mdb"
Set db = OpenDatabase(strFileName)
```

מסד נתונים מרוחק

השירות **OpenDatabase** מיועד לפניה למסד נתונים מקורב, אך כאשר נרצה לפנות למסד נתונים מרוחק (כמו למשל מסד נתונים של Oracle שמותקן בשרת), ניעזר בסוג המשתנה **rdoConnection** שבעזרתו ניצור את הקשר אל מסד הנתונים. משתנה נוסף שדרוש הוא **rdoEnvironment**, שעליו נפעיל את השירות **OpenConnection** ליצירת הקשר אל מסד הנתונים המרוחק. לדוגמה:

```
Private mEn As rdoEnvironment
Private mConn As rdoConnection

Set mEn = rdoEngine.rdoEnvironments(0)

'connection to remote database:
Set mConn = mEn.OpenConnection _
("Oracle", rdDriverNoPrompt, False, "UID=UserId;PWD=Password;")
```

השירות OpenConnection מקבל ארבעה פרמטרים עיקריים.

1. **Data Source Name** – DSN הוא השם בו מיוצג מסד הנתונים ב- ODBC.
2. **Prompt** – קבוע המייצג את האופן בו הפעולה תצא אל הפועל.
3. **ReadOnly** – משתנה בוליאני הקובע אם מסד הנתונים ייפתח לקריאה בלבד או לא.
4. **Connect** – מחרוזת המועברת ל- ODBC לפתיחת מסד הנתונים. המחרוזת מקבלת את User Id (שם המשתמש) ו-Password (סיסמת הכניסה למסד הנתונים). שים לב שבדוגמה המחרוזת בקוד מכילה את שם המשתמש והסיסמה. במצב זה מסד הנתונים ייפתח אוטומטית. אולם ישנה אפשרות לא לשלוח את הסיסמה, ואז המשתמש יצטרך להקיש אותה בעצמו, כדי שיורשה להיכנס למסד הנתונים.

המשתנה mConn מייצג את הקשר אל מסד הנתונים המרוחק, ומעתה כל פנייה למסד הנתונים תהיה דרך משתנה זה. לדוגמה:

```
Dim rsQuery As rdoResultset
Dim strQuery As String

strQuery = "SELECT Id, LastName, FirstName FROM Employees"
Set rsQuery = mConn.OpenResultset(Name:=strQuery, _
                             Type:=rdOpenForwardOnly, _
                             LockType:=rdConcurReadOnly, _
                             Options:=rdExecDirect)
```

ברוב השירותים שנלמד בהמשך (מלבד אלו המיוחדים לאובייקט RDO) ניתן להשתמש גם במקרה בו אנו פונים למסד נתונים מרוחק.

פתיחת טבלה

רוב פעולות ניהול מסד הנתונים נעשות בעיקר מול טבלה אחת, או מול מספר טבלאות. עיקר הפעולות הן שליפת נתונים, כתיבה, עריכה ומחיקה. כדי לפנות לטבלה במסד הנתונים יש להצהיר על משתנה מתאים ולקשר אותו עם הטבלה. הקשר בין המשתנה לטבלה נעשה בעזרת השירות **OpenRecordset**, כאשר הפרמטר שמקבל השירות הוא שם הטבלה במסד הנתונים.

לכל טבלה דרוש משתנה נפרד, אלא אם הקשר בין המשתנה לטבלה הראשונה הסתיים ובמקומו בא קשר עם טבלה אחרת.

תחביר השירות הוא:

```
Set variable = Database.OpenRecordset(Table name)
```

לדוגמה:

```
Dim db As Database
Dim r As Recordset
Set db = OpenDatabase("c:\Nwind.mdb")
Set r = db.OpenRecordset("Orders")
```

השירות מקבל פרמטרים נוספים, אולם לא נתייחס אליהם במסגרת ספר זה. בנוסף, במקום שם הטבלה ניתן לשלוח כפרמטר משפט SQL ועל כך נדון בהמשך.

הפניה לשדה בטבלה

ברגע שנוצר הקשר בין המשתנה לבין הטבלה באמצעות השירות OpenRecordset ניתן לפנות לשדות בטבלה על פי התחביר הבא:

```
Variable![Field name]
```

לדוגמה:

```
Dim db As Database
Dim r As Recordset
Set db = OpenDatabase("c:\Work.mdb")
Set r = db.OpenRecordset("Employees")

Dim strName As string
Dim intAge As Integer
strName = r![FirstName]
intAge = r![Age]
```

שים לב שלאחר הפעלת השירות OpenRecordset מצביע הקלט מוצב על הרשומה הראשונה. פקודות ההשמה הבאות מכניסות את הערכים הנמצאים בשדות FirstName ו-Age שברשומה הראשונה אל המשתנים StrName ו-intAge, בהתאמה:

```
strName = r![FirstName]
intAge = r![Age]
```

השירות OpenRecordset עם משפט SQL

השירות OpenRecordset מקבל בדרך כלל כפרמטר שם טבלה. אולם ניתן לשלוח כפרמטר גם משפט SQL. במקרה זה, ההתייחסות תהיה אל הרשומות שהתקבלו ואל השדות הכלולים בשאילתה. שים לב שייתכן שלא נקבל את כל הרשומות הקיימות בטבלה, אלא רק את חלקן, על פי תחביר השאילתה (משפט SQL). לדוגמה:

```
Dim db As Database
Dim r As Recordset
Dim strSql As String
Set db = OpenDatabase("c:\Work.mdb")
strSql  = "Select FirstName , Age From Employees"
Set r = db.OpenRecordset(strSql)
```

בדוגמה זו השירות מתייחס לתוצאות השאילתה ולשדות הכלולים במשפט SQL בלבד, ולא לכל שדות הטבלה. בדוגמה לעיל ההתייחסות תהיה לכל הרשומות בטבלה Employees, אך לשדות FirstName ו-Age בלבד.

הפניה לשדה בשאילתה

בדומה לשדה בטבלה, הפניה לשדה בשאילתה נעשית על פי התחביר הבא:

```
variable![field]
```

לדוגמה:

```
Dim strName As string
strName = r![FirstName]
```

לא ניתן לפנות לשדה Id שבטבלה Employees מכיון שהוא אינו חלק ממשפט SQL.

אפשרות נוספת לפנות לשדה בשאילתה היא בציון מיקומו הכרונולוגי במשפט SQL. לדוגמה:

```
Dim strName As String
Dim intAge As integer
strName = r(0)
intAge = r(1)
```

r(0) מתייחס לשדה FirstName שהוא הראשון בשאילתה ו-r(1) מתייחס לשדה Age שנמצא אחריו. שים לב שהיתרון הגדול שבשימוש במשפט SQL כפרמטר לשיטה OpenRecordset הוא ביכולת לפעול על מספר טבלאות. בניגוד לשימוש הרגיל שבו הפרמטר מתייחס לטבלה אחת, במשפט SQL אחד ניתן לקבל מידע משדות במספר טבלאות. לדוגמה:

```
Dim strSql As String
strSql = "Select Employees.FirstName , Employees.Age , " & _
       "Department.Manager From Employees , Department " & _
       "Where Employees.DepId = Department.DepId"
Set r = db.OpenRecordset(strSql)
```

בדוגמה זו, r(0) ו-r(1) יתייחסו לשדות FirstName ו-Age מטבלת Employees בהתאמה, ואילו r(2) יתייחס לשדה Manager המקביל להם מטבלת Department.

עריכת רשומות

עסקנו עד עתה בפתיחת מסד הנתונים ובקישור השירות OpenRecordset לאחת מטבלאותיו. גם למדנו על הפניה לשדה בעזרת משתנה מסוג Recordset לשליפת מידע מהטבלה. השימוש במשתנה מסוג זה והפניה דרכו אל שדות הטבלה אינה לאחזור נתונים בלבד, כי אם לעריכת רשומות הטבלה. עריכת רשומות בטבלה נעשית בעזרת שירות מתאים, שבה הפניה לשדה מורה לשירות על איזה שדה או על איזו רשומה לבצע את העריכה.

השירות AddNew

חלק מניהול הטבלה במסד נתונים הוא היכולת להוסיף רשומה חדשה לטבלה. השירות המורה לטבלה לפתוח רשומה חדשה ולקבל שדות חדשים הוא **AddNew**. לאחר הפעלת השירות AddNew, פקודות ההשמה תבוצענה על שדות הטבלה. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.AddNew
r![FirstName] = "Danny"
r![Age] = 21
r![ID] = 12345
```

אין חובה לבצע פעולות השמה ולהכניס ערכים לכל שדות הטבלה וניתן להשאיר שדות מסוימים ריקים מערכים. כל זאת, בתנאי שהשדה מאפשר קבלת ערך ריק והוא אינו שדה חובה.

בהפעלת השירות **AddNew** אין כל משמעות באיזו רשומה נמצאים בעת הפעלתו. בכל מקרה הרשומה תתוסף בסוף הטבלה.

השירות Edit

באופן דומה לשירות **AddNew** המאפשר הוספת רשומה חדשה לטבלה, השירות **Edit** מאפשר עריכת רשומה קיימת. כאשר התוכנית קוראת לשירות Edit, כל פקודות ההשמה הבאות עורכות את הרשומה הנוכחית. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.Edit
r![FirstName] = "Yossy"
r![Age] = 22
r![ID] = 56789
```

שים לב שהרשומה הנוכחית היא הרשומה הראשונה והשירות Edit יפעל עליה. כלומר, הערכים החדשים יעודכנו ברשומה הראשונה, ובה בלבד.

השירות Update

אם נוסיף רשומה חדשה בעזרת השירות AddNew או אם נערוך רשומה קיימת בעזרת השירות Edit, הפעולות תבוצענה בשטח העבודה בלבד ללא עדכון הטבלה שבדיסק. השינויים ייכתבו בטבלה **רק** על ידי השירות Update. לפי זה, לאחר כל שימוש בשירות AddNew או בשירות Edit חובה להפעיל את השירות Update כדי לגרום לשינוי בטבלה עצמה. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")

r.Edit
r![FirstName] = "Moshe"
r![DepCode] = 1
r.Update

r.AddNew
r![FirstName] = "Avi"
r![Age] = 31
r![ID] = 12271
r![DepCode] = 3
r.Update
```

בדוגמה זו מתבצעת עריכת הרשומה הראשונה ועדכונה בטבלה; לאחר מכן יש הוספת רשומה חדשה ומילוי השדות בערכים ולבסוף – עדכון הטבלה. שים לב שלכל שירות מתבצע עדכון הטבלה בנפרד.

השירות Delete

השירות **Delete** מוחק את הרשומה הנוכחית. אם מצביע הקלט מוצב ברשומה הראשונה ובשלב זה מתבצעת הקריאה לשירות, שורה זו נמחקת. אם הרשומה הנוכחית היא האחרונה – שורה זו תימחק, וכן הלאה. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Department")
r.MoveLast
r.Delete
```

בדוגמה זו מוחקים את הרשומה האחרונה. ניתן לראות שאין צורך לבצע עדכון באמצעות השירות Update: השירות Delete גורם בעצמו למחיקת הרשומה מהטבלה.

דפדוף ברשומות

בכל פעם שמבוצע המשפט הבא:

```
Set r = db.OpenRecordset
```

השירות **OpenRecordset** גורם למצביע הקלט לעמוד ברשומה הראשונה של הטבלה. אם נבצע עדכון או מחיקה, פעולות אלו תתייחסנה לרשומה הנוכחית ובמקרה זה – הראשונה. לרוב לא נרצה לעדכן או למחוק את הרשומה הראשונה דווקא, אלא רשומה אחרת כלשהי. במקרה זה עלינו לדפדף בין הרשומות לפנים ולאחור עד לרשומה המבוקשת. כאשר נעמוד ברשומה זו, כל קריאה לשירות עריכה או מחיקה יתבצע עליה, כי זו הרשומה הנוכחית המוגדרת.

השירות MoveFirst

השירות **MoveFirst** מציב את מצביע הקלט ברשומה הראשונה. אם מצביע זה נמצא כבר ברשומה זו, לא תהיה משמעות לביצוע פעולה זו ואין זו שגיאה. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Department")
r.MoveFirst
```

בדוגמה זו לא נעשה שינוי כתוצאה מהקריאה לשירות, מכיון שהמצביע כבר מוצב ברשומה הראשונה.

השירות MoveLast

שירות זה מציב את המצביע ברשומה האחרונה. כל שינוי שנעשה ברשומה לאחר הקריאה לשירות MoveLast יתייחס לרשומה האחרונה. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.MoveLast
r.Delete
```

בדוגמה זו, הרשומה שתימחק היא הרשומה האחרונה.

השירות MoveNext

קריאה לשירות **MoveNext** גורמת למצביע לעמוד על הרשומה הבאה לאחר הרשומה הנוכחית בו הוא נמצא כעת. אם לדוגמה קראנו לשירות MoveFirst ואחריו לשירות MoveNext, הרשומה הנוכחית היא הרשומה השנייה. בדוגמה זו השירות מצביע על הרשומה השנייה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.MoveNext
r.MoveNext
```

ראינו שבשירותים MoveFirst ו-MoveLast, אין מצב שגיאה כאשר המצביע נמצא ברשומה הראשונה או ברשומה האחרונה, בהתאמה, ולא יתבצע כל שינוי. לעומת זאת, השירות MoveNext יכול לגרום לשגיאה אם המצביע מורה על הרשומה האחרונה בעת שמבצעים אותו. מכיון שאין רשומות נוספות, כביכול חרגנו מתחום הטבלה. כדי להימנע משגיאה זו ניעזר במאפיין EOF.

המאפיין EOF

מאפיין זה מסייע לגלות אם הגענו לרשומה האחרונה או לא. כאשר מצביע הקלט מצביע על הרשומה האחרונה, המאפיין **EOF** מקבל ערך True, בכל רשומה אחרת ערך המאפיין הוא False. לכן, בכל פעם שנתקדם רשומה אחת בעזרת השירות MoveNext, נבדוק תחילה את ערך המאפיין EOF. אם ערכו False, הרשומה הנוכחית אינה האחרונה ונוכל לקרוא לשירות MoveNext. אם ערכו True – הרשומה הנוכחית היא האחרונה ואין להתקדם.

השירות MovePrevious

השירות **MovePrevious** גורם להזזת המצביע רשומה אחת לאחור מהרשומה הנוכחית שבה הוא מוצב. כל קריאה נוספת לשירות מזיזה את המצביע רשומה אחת לאחור. לדוגמה:

```
Dim r AS Recordset
Set r = db.OpenRecordset("Employees")
r.MoveLast
r.MovePrevious
```

בדוגמה זו, לאחר ביצוע המשפט Set המצביע מוצב על הרשומה הראשונה. כאשר השירות MovePrevious נקרא, הוא גורם למצביע לזוז רשומה אחת לאחור. שים לב, אם המצביע היה מוצב ברשומה הראשונה, הזזתו לאחור תגרום לחריגה מתחום הרשומות ולהודעת שגיאה שלא קיימת רשומה במקום בו המצביע נמצא. כדי למנוע תקלה זו ניעזר במאפיין **BOF**.

המאפיין BOF

המאפיין **BOF** מסייע לגלות אם הרשומה הנוכחית היא הראשונה בטבלה. כאשר המצביע מוצב על הרשומה הראשונה, המאפיין BOF מקבל ערך True וכאשר הוא מוצב על כל רשומה אחרת ערכו False. בכל פעם שנרצה לקרוא לשירות MovePrevious נבדוק תחילה את ערך המאפיין BOF. כאשר ערכו False נוכל לקרוא לשירות MovePrevious, אך כאשר ערכו True לא נוכל לעשות זאת. לדוגמה:

```
Private Sub cmdPrevious_Click()
   If Not r.BOF Then r.MovePrevious
End Sub
```

השירות Move

השירותים שסקרנו עד כה סיפקו אפשרות לנוע לרשומה מוגדרת, הראשונה, האחרונה, רשומה אחת קדימה ורשומה אחת לאחור. שירות נוסף מאפשר תנועה יותר חופשית ברשומות. שירות זה הוא **Move**, המאפשר להגדיר את מספר הצעדים שנרצה להזיז את המצביע לפנים או לאחור. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Select, Id FirstName From Employees")
r.Move 3
Print r![FirstName]
```

בדוגמה זו, המצביע ינוע אל הרשומה הרביעית, שלושה צעדים קדימה מהרשומה הראשונה. כדי להזיז את המצביע לאחור, מספר הצעדים שיקבל השירות כפרמטר יהיה שלילי. לדוגמה:

```
r.Move -2
Print r![ID]
```

שים לב, שחריגה מתחום הרשומות יוצרת מצב שגיאה. כאשר המצביע מוצב על הרשומה הראשונה, מספר הצעדים המקסימלי שהוא יכול להתקדם לאחור הוא: RecordCount -1.

המאפיין RecordCount

המאפיין **RecordCount** מייצג את מספר הרשומות בטבלה שפתחנו בעזרת השירות OpenRecordset. לדוגמה,

```
Dim r As Recordset
Set r = db.OpenRecordset("Department")
Print r.RecordCount
```

אם בטבלה Department יש 20 רשומות, התוכנית תדפיס את המספר 20.

שים לב שהמאפיין RecordCount אינו מייצג את מספר הרשומה הנוכחית, אלא כמה רשומות יש בכל הטבלה שהביא השירות OpenRecordset.

ניתן להשתמש במאפיין RecordCount כדי להדפיס את כל הרשומות. מכיון שמאפיין זה מייצג את מספר כל הרשומות, ניתן להריץ לולאה שתתחיל למנות מהרשומה הראשונה ועד RecordCount. לדוגמה:

```
Dim I As Integer
For I= 1 To r.RecordCount
    Print r![Manager]
    r.MoveNext
Next I
```

ניתן להשתמש במאפיין EOF כדי לקבל את אותה תוצאה. נורה ללולאה לפעול מהרשומה הראשונה ועד לרשומה האחרונה ונבחן את המאפיין EOF כדי לדעת אם הגענו לרשומה האחרונה או לא. לדוגמה:

```
r.MoveFirst
Do While Not r.EOF
   Print r![Manager]
   r.MoveNext
Loop
```

באופן דומה ניתן לפעול בלולאה מהרשומה האחרונה לכיוון הרשומה הראשונה, אך הפעם, המאפיין שנסתייע בו הוא BOF. לדוגמה:

```
r.MoveLast
Do While Not r.BOF
   Print r![Manager]
   r.MovePrevious
Loop
```

המאפיין RecordCount במשפט SQL

השירות OpenRecordset יכול לקבל כפרמטר שם טבלה, אך ראינו כי ניתן לו גם משפט SQL כארגומנט. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Select Manager From Department")
```

תוצאת הקריאה לשירות תהיה רשימה שתכיל את שמות כל המנהלים שבטבלה Departments. נניח כי אנו רוצים לדעת כמה רשומות נתקבלו, נפנה למאפיין RecordCount ונדפיס את ערכו. לדוגמה:

```
Dim r As Recordset
Dim strSql As String
strSql = "Select Manager From Department"
Set r = db.OpenRecordset(strSql)
Print r.RecordCount
```

התוצאה שתודפס תהיה 1. כלומר, הערך שמכיל המאפיין RecordCount הוא 1! הסיבה לכך היא, שהמאפיין RecordCount במקרה זה אינו מייצג את מספר כל הרשומות, אלא את הרשומה הנוכחית שעליה הוא מוצב. במילים אחרות, יש הבדל בין מה שמייצג המאפיין RecordCount כאשר השירות OpenRecordset נקרא עם טבלה כפרמטר, לבין התוצאה המתקבלת כאשר אותו שירות מקבל משפט SQL כפרמטר.

כאשר הפרמטר הוא טבלה, המאפיין RecordCount מייצג את מספר כל הרשומות; וכאשר הפרמטר הוא משפט SQL, המאפיין מחזיר כערך את מספר הרשומה הראשונה, היינו 1. ערך זה מתקבל כאשר מצביע הקלט נמצא ברשומה הראשונה ועדיין לא בוצעה כל פעולה שהיא על תוצאת השירות. לכן, לפני בדיקת ערכו של המאפיין נקרא לשירות MoveLast, כדי לעדכן את המאפיין RecordCount במספר כל הרשומות שהתקבלו. לדוגמה:

```
Dim r AS Recordset
Dim strSql As String
strSql = "Select Manager From Department"
Set r = db.OpenRecordset(strSql)
```

```
r.MoveLast
Print r.RecordCount
```

במקרה זה התוצאה שתודפס תהיה מספר כל הרשומות שהתקבלו מהשאילתה.

איתור רשומות

עד כה עסקנו בדפדוף רשומות קדימה ואחורה, לכיוון הרשומה האחרונה, או אל כיוון הרשומה הראשונה, בהתאמה. אך מה נעשה אם נרצה להגיע לרשומה מסוימת, נניח אל הרשומה החמישית? במקרה כזה, כאשר המצביע מוצב על השורה הראשונה בטבלה, עלינו לקרוא 4 פעמים לשירות MoveNext בעזרת לולאה, כדי שיציב את המצביע על הרשומה החמישית. אפשרות אחרת היא לעמוד על הרשומה הראשונה ולהתקדם 4 צעדים. אך מה נעשה כשנרצה שהמצביע יעמוד על הרשומה של העובד "דני" שמספר ת"ז שלו 12345? במקרה זה אין אנו יודעים באיזה מספר רשומה נמצאים הנתונים על עובד זה. לשם כך ניעזר במאפיינים של המשתנה מסוג Recordset כדי למצוא רשומות מסוימות בטבלה המקורית או בתוצאה שהפיקה השאילתה.

המאפיין Index

המונח **אינדקס** (Index) מוכר לנו. כאשר יצרנו טבלה והוספנו לה שדות הגדרנו חלק מהשדות כאינדקסים. גם ראינו שאין חובה ליצור אינדקס בטבלה, אך במקרים מסוימים הימצאות האינדקס מסייעת מאוד. מטרת האינדקס היתה חיפוש מהיר של רשומה מכיון שהרשומות נשמרות בטבלת האינדקס באופן ממוין. כל טבלת אינדקס מקבלת שם שאליו אנו פונים כדי להשתמש באותו אינדקס (אין לבלבל בין טבלת האינדקס, שהיא למעשה טבלת שירות פנימית, לבין טבלת הנתונים שלנו).

נניח לדוגמה, שבטבלה **Employees** יצרנו אינדקס מפתח לשדה **Id**. כאשר נרצה לפנות ישירות לרשומה של העובד דני בעל ת"ז 12345, נפנה בשלב ראשון לאינדקס (במקרה זה, למפתח) המכיל את השדה Id:

```
Recordset.Index = indexname
```

לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("MyTable")
r.Index = "My Index"
```

במקרה שלנו האינדקס הוא גם מפתח, וכבר ראינו שהמפתח נקרא גם **PrimaryKey**. בכל פעם שנפנה לאינדקס שהוא מפתח, נפנה אליו בשם זה (זכור, בטבלה אחת יכולים להיות כמה אינדקסים, אך מפתח **אחד** בלבד). שורות הקוד המתאימות ייראו כך:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.Index = "PrimaryKey"
```

משפנינו לאינדקס המתאים, נשאר רק להורות לו איזו רשומה עליו לחפש.

המאפיין Seek

המאפיין Seek מאתר רשומה מסוימת בתוך אינדקס.

כך כותבים את המשפט:

```
recordset.Seek comparison, Key1, Key2,.....Key13
```

כאשר,

recordset - מייצג משתנה מסוג recordset.

Comparison - מייצג מחרוזת המייצגת אחד מאופרטורי ההשוואה הבאים: <, =<, >=, >, =.

Key - מייצג את הערך בשדה האינדקס שאותו יש למצוא. ניתן לבנות עד 13 ערכים, כאשר האינדקס כולל 13 שדות.

שים לב שישנה התאמה בין מספר ה-Key לבין מספר השדות שבאינדקס. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.Index = "PrimaryKey"
r.Seek "=", 12345
```

בדוגמה זו יש פנייה למפתח (המכיל את השדה Id), והתוכנית מבקשת ממנו למצוא את הרשומה שהשדה Id שלה שווה ("=") למספר 12345. האינדקס מוצא את השדה המתאים ופונה ישירות לרשומה המתאימה לו בטבלת Employees ושולף ממנה את הנתונים.

כאשר נרשום את המשפט הבא:

```
Print r![FirstName]
```

נקבל את השם "דני" שחיפשנו, מכיון שהאינדקס איתר על פי ת"ז 12345 את המצביע אל הרשומה המתאימה בטבלת הנתונים. כל פנייה לשדה ברשומה זו תספק את הנתונים הדרושים. אם לא נמצאה רשומה העונה לתנאי החיפוש, משפט זה יגרום לתוכנית לעצור עם הודעת שגיאה "No current record". כדי למנוע זאת, ניעזר במאפיין **NoMatch**.

המאפיין NoMatch

המאפיין **NoMatch** מציין אם רשומה שחיפשנו בעזרת Seek נמצאה או לא. אם הרשומה לא נמצאה, ערך המאפיין הוא True. לגבינו, כאשר אנו מבצעים חיפוש רשומה באינדקס בעזרת המאפיין Seek, עלינו לוודא תחילה בעזרת NoMatch אם הרשומה אכן נמצאת או לא. אם הרשומה לא נמצאה, אין לפנות למשתנה Recordset ולבקש ממנו נתונים. לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Employees")
r.Index = "PrimaryKey"
r.Seek "=", 12345
If Not NoMatch Then
   Print r![ID]
   Print r![FirstName]
End If
```

בדוגמה התנינו את הדפסת השדות Id ו-FirstName בערך המאפיין NoMatch. רק כאשר ערכו False (כלומר, הרשומה נמצאה), התוכנית מדפיסה את ערכי השדות האלה.

השירות FindFirst

השירות Seek המשולב עם המאפיין Index מסייע למצוא רשומה מוגדרת כלשהי. שירות זה והמאפיין Index תקפים רק כאשר השירות OpenRecordset מופעל על טבלה. כאשר השירות מופעל על משפט SQL עלינו להיעזר בחיפוש רשומה לפי ערכה, בעזרת השירות **FindFirst**. השירות FindFirst מוצא את הרשומה הראשונה שמתאימה לקריטריון החיפוש שהגדרנו.

תחביר הקריאה לשירות:

```
recordset.FindFirst criteria
```

כאשר,

recordset - משתנה מסוג Recordset.

Criteria - מחרוזת (או משתנה String) המכילה את הקריטריון. מחרוזת הקריטריון כתובה באופן דומה לחלק "Where" במשפט SQL, אך ללא המילה "Where". לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Select FirstName, Id From Employees")
r.FindFirst "Id = 12345"
Print r![FirstName]
```

שים לב שהשירות FindFirst מתחיל את החיפוש ברשומה הראשונה, וכיוון החיפוש הוא כלפי הרשומה האחרונה.

השירות FindNext

השירות **FindNext** מוצא את הרשומה הבאה שמתאימה לקריטריון החיפוש שהוגדר. גם שירות זה מקבל מחרוזת כקריטריון הכתוב באופן דומה לחלק "Where". לדוגמה:

```
Dim r As Recordset
Set r = db.OpenRecordset("Select FirstName, Id From Employees")
```

```
r.FindFirst "Id = 12345"
Print r![FirstName]
r.FindNext "Id = 12345"
Print r![FirstName]
```

שים לב, השירות FindNext מתחיל את החיפוש ברשומה הנוכחית לכיוון הרשומה האחרונה.

השירות FindLast

שירות זה מסייע בחיפוש רשומות על פי קריטריון מתוך הרשומות החוזרות כתשובה לשאילתה. השירות מוצא את הרשומה האחרונה שמתאימה לקריטריון החיפוש שהוגדר. נניח לדוגמה, שנתונות הרשומות שבטבלה 16.1 :

טבלה 16.1

DepId	Manager
1	Danny
2	Avi
3	Danny
4	Moshe

שורות הקוד הבאות :

```
Dim r As Recordset
Set r = db.OpenRecordset("Select DepId, Manager From
Department")
r.FindLast "Manager ='Danny'"
Print r![DepId]
```

התוצאה שנקבל: 3. כלומר, הרשומה השלישית היא האחרונה מבין הרשומות שתואמות את תנאי החיפוש.

השירות FindLast מתחיל את החיפוש מהרשומה האחרונה לכיוון הרשומה הראשונה.

השירות FindPrevious

השירות **FindPrevious** מוצא את הרשומה הקודמת שמתאימה לתנאי הקריטריון. שירות זה מתחיל את החיפוש מהרשומה הנוכחית ונע אחורה לכיוון הרשומה הראשונה. אם נתייחס לנתונים שבטבלה 16.1 ולשורות הקוד הקודמות, שורות הקוד האלו :

```
r.FindPrevious "Manager = 'Danny'"
Print r![DepId]
```

ידפיסו את התוצאה : 1. כלומר, הרשומה הראשונה (1) היא הקודמת.

טרנזקציות

נניח שאנו מנהלים מסד נתונים בו נתונות שתי טבלאות, טבלת מלאי וטבלת מכירות. בטבלת ה**מלאי** נשמר המידע על כמות המוצרים הקיימים במלאי, ובטבלת ה**מכירות** נשמר המידע על כמות המוצרים שנמכרו. המשתמש מצפה שבזמן שהוא ממלא הזמנה ללקוח, הכמות שבהזמנה תופחת אוטומטית בטבלת המלאי ותעדכן את טבלת המכירות. בצורה זו יידעו מנהלי המלאי בדיוק כמה נשאר במלאי ומתי צריך להזמין סחורה לחידוש המלאי. לאור הדוגמה, בחן את שורות הקוד הבאות:

```
Dim intQuantity As Integer
Dim recOrder As Recordset
Dim recStock AS Recordset
```

המשתנה intQuantity מאחסן את כמות המוצרים שנמכרו.

המשתנה recOrder מנהל את טבלת המכירות.

המשתנה recStock מנהל את טבלת המלאי.

```
IntQuantity = Val(frmSale.txtSale.Text)
Set recOrder = db.OpenRecordset("Orders")
Set recStock = db.OpenRecordset("Stock")
```

עדכון טבלת המכירות (הוספת רשומה חדשה עם נתוני המכירה):

```
recOrder.AddNew
recOrder![ProdactName] = frmSale.txtProdact.Text
recOrder![Quantity] = intQuantity
recOrder.Update
```

השלב השני עדכון טבלת המלאי (הורדת כמות המכירה מכמות המלאי הקיימת):

```
recStock.Index = "ProdactId"
recStock.Seek "=", Val(frmSale.txtId.Text)
recStock.Edit
recStock![Quantity] = recStock![Quantity] - intQuantity
recStock.Update
```

עד כאן הכל טוב ויפה, טבלת המכירות עודכנה בכמות ההזמנה וטבלת המלאי עודכנה בהתאם ומנהלי המלאי יוכלו לראות שמהמלאי ירדה כמות מסוימת של מוצרים. אך מה יקרה אם בין השלב הראשון (עדכון טבלת המכירות) לבין השלב השני (עדכון טבלת המלאי) קרתה הפסקת חשמל. בפועל, טבלת המכירות עודכנה על פי כמות המוצרים שהוזמנו, אך טבלת המלאי לא עודכנה בהתאם. מנהלי המלאי לא יידעו שבוצעה הזמנה והמלאי לא הופחת.

כדי להתגבר על תקלה זו ולמנוע אותה מראש, "נעטוף" את כל התהליך כ**תנועה** (transaction). משמעות הדבר היא ששני שלבי התהליך, הכוללים את עדכון טבלת המכירות ואת עדכון טבלת המלאי, יחשבו תהליך **אחד**. כאשר התהליך נפסק באמצע מסיבה כלשהי, התוכנית לא תעדכן את חלק התהליך שכבר בוצע. כלומר, כל עוד לא

הגיע התהליך לסיומו הכולל במקרה זה את עדכון טבלת המלאי, גם עדכון המכירות לא יבוצע.

לפניך שורות הקוד ה"עוטפות" את התהליך כולו כ**תנועה**:

```
Dim wspProg As Workspace
Dim dbSale As Database
Dim recOrder As Recordset
Dim recStock As Recordset
Dim intQuantity As Integer

Set wspProg = DBEngine.Workspaces(0)
Set dbSale = wspProg.OpenDatabase("c:\Sales.mdb")
Set recOrder = dbSales.OpenRecordset("Orders")
Set recStock = dbSales.OpenRecordset("Stock")
intQuantity = Val(frmSale.txtSale.Text)

wspProg.BeginTrans

recOrder.AddNew
recOrder![ProdactName] = frmSale.txtProdact.Text
recOrder![Quantity] = intQuantity
recOrder.Update

recStock.Index = "ProdactId"
recStock.Seek "=", Val(frmSale.txtId.Text)
recStock.Edit
recStock![Quantity] = recStock![Quantity] - intQuantity
recStock.Update

wspProg.CommitTrans
```

התהליך כולו עטוף כתנועה המתחילה ב-wspProg.BeginTrans ומסתיימת ב-wspProg.CommitTrans. כאשר תהיה תקלה במהלך העבודה, התהליך **לא** יעדכן את מסד הנתונים והפעולה תתבטל. לדוגמה, אם תהיה הפסקת חשמל לאחר עדכון טבלת המכירות ולפני עדכון המלאי, התוכנית לא תגיע לשורה wspProg.CommitTrans ולכן – כל התהליך יבוטל. פירוש הדבר שטבלת המכירות לא תעודכן והטבלאות תישארנה ללא שינוי.

השירות RollBack

השירות **BeginTrans** מתחיל את הפעולות הכלולות בתנועה, והשירות **CommitTrans** מסיים אותה. כשהתוכנית קוראת לשירות CommitTrans הוא מעדכן את כל התהליך שכלול בתנועה. עם זאת, אפשר לבטל את התהליך ולהפסיק בכל עת את הפעולות שבוצעו על ידי התנועה.

כשנפעיל את השירות CommitTrans הוא יבצע את חלק התהליך שכבר בוצע. לפעמים נרצה **לבטל** את כל התהליך ולשחזר את המצב לקדמותו, ולצורך זה נפעיל את השירות **RollBack**. השירות RollBack מבטל את כל התהליך שבוצע לאחר הקריאה לשירות BeginTrans ומחזיר את מסד הנתונים למצב שהיה לפני הפעלת השירות BeginTrans. לדוגמה:

```
wspProg.BeginTrans
      Progress...
wspProg.RollBack
```

שים לב: אם בוצעה קריאה לשירות CommitTrans, התהליך שכלול בתנועה מבוצע ולקריאה לשירות RollBack לא תהא כל משמעות וגם תהיה שגיאה. לאחר הקריאה לשירות BeginTrans התוכנית מצפה לשירות CommitTrans לאישור התנועה או לשירות RollBack - לביטולה. כאשר אחד מהשירותים הופעל, קריאה לשירות השני, ללא קריאה נוספת לשירות BeginTrans, תגרום מצב שגיאה.

ניהול מסד נתונים בזמן פעולת התוכנית

פיתוח יישום ללקוח כולל תחילה את אפיון דרישותיו על ידי מנתח המערכת. המתכנת מתרגם אפיון זה למסד נתונים ולממשק משתמש, שהוא יישום וויז'ואל בייסיק, כדי לנהל מסד נתונים זה. לאחר שהיישום נמסר ללקוח והוא משתמש בו, קורה שהוא דורש שינויים נוספים שלא חשב עליהם קודם. במקרה בו השינויים הם ברמת הניהול בלבד, עבודת המתכנת פשוטה. אולם לפעמים הלקוח דורש שינוי מבני במסד הנתונים, כמו למשל הוספת שדה לטבלה קיימת. לדוגמה, הלקוח מחליט שהוא מקצה רכבים לחלק מהמנהלים בחברה, והוא מעוניין בשדה נוסף בטבלת **מנהלים**, שבו יישמר ברשומת כל מנהל מידע בדבר רכב שקיבל, אם קיבל. בנוסף, בטבלה **רכבים** (בהנחה שהיא נמצאת מסד הנתונים) יושלמו בכל רשומת רכב פרטי המנהל שקיבל את הרכב לשימושו. מלבד זאת, הלקוח עשוי לדרוש דוחות שונים המציגים מידע על הרכבים בחתכים שונים, דרישה שבעקבותיה ייתכן שיהיה צורך להוסיף לטבלה אינדקס.

יוצא מכך, שהמבנה הראשון של מסד הנתונים שבנינו אינו מספק אותנו עוד. לפעמים נצטרך לנהל מסד נתונים מבחינה פונקציונלית ומבנית בזמן הפעלת התוכנית, כדי להתאימו לצרכים משתנים.

לביצוע שינויים מבניים במסד הנתונים בזמן ריצה קיימות שתי דרכים עיקריות. במסגרת ספר זה נדון בדרך הקלה יותר, אך גם זו דורשת ידע בסיסי בשפת SQL. אל דאגה, את משפטי SQL הנדרשים לביצוע נציג בפירוט לאלה שאינם מכירים אותם.

השירות Execute

השירות **Execute** מבוצע על משתנה מסוג **Database**. שירות זה מבצע משפט SQL אשר פועל על מסד הנתונים שמייצג משתנה מסוג Database שעליו הופעל השירות.

תחביר הקריאה לשירות:

```
database.Execute SQLstatement
```

את משפט SQL ניתן להחליף במשתנה String המייצג גם הוא משפט SQL. שים לב שמשפט SQL זה מייצג שאילתת ביצוע ולא שאילתת בחירה. לדוגמה:

```
Dim db As Database
Set db = OpenDatabase("c:\MyWork.mdb")
db.Execute "Update Employees Set FirstName ='Avi'" & _
                  "Where Id = 12345"
```

ואפשר לכתוב זאת גם כך:

```
Dim db As Database
Dim strSql As String
Set db = OpenDatabase("c:\MyWork.mdb")
strSql = "Update Employees Set FirstName = 'Avi'"
strSql = strSql & " Where Id = 12345"
db.Execute strSql
```

השילוב של שאילתת ביצוע ושל קריאה לשירות Execute מאפשר לבצע כל שינוי מבני במסד הנתונים. כאשר נרצה להוסיף שדה לטבלה, נתרגם את בקשתנו למשפט SQL ונשלח אותו כארגומנט לשירות Execute.

הוספת טבלה

כדי להוסיף טבלה למסד הנתונים בזמן הפעלת התוכנית, נכתוב משפט SQL המורה על הוספת טבלה ונקרא לשירות Execute עם משפט זה.

למרות ששפת SQL היא שפה סטנדרטית, נמצא לפעמים שינויים בתחביר השפה בין גירסאות שונות של השפה. בסעיפים הבאים נתייחס לשפת SQL אשר מיושמת במערכת ניהול מסדי הנתונים **Microsoft Access** ו-**Oracle**. כאשר יהיה שוני ביניהם נציין זאת במפורש.

תחביר משפט SQL להוספת טבלה הוא:

```
Create Table table_name (Field_name data_type  (size),
Field_name data_type)
```

לדוגמה:

```
Create Table curs (Id Text (10), Manager Text (20), Price
Integer)
```

משפט SQL יכול להיכתב באותיות קטנות או גדולות (באנגלית), כי אין לכך משמעות. את גודל שדה ניתן להגדיר ב-Access בשדה מסוג Text או Binary בלבד.

סוג השדה Text מקביל לסוג המשתנה String בוויז'ואל בייסיק, וכך הוא מוגדר במסד הנתונים Access. ב-Oracle מוגדר אותו סוג שדה כ-VarChar.

מחיקת טבלה

תחביר משפט SQL למחיקת טבלה הוא:

```
Drop Table table_name
```

לדוגמה:

```
Drop Table Curs
```

כשנרצה למחוק טבלה ממסד הנתונים, נקרא לשירות Execute עם משפט SQL זה.
לדוגמה:

```
db.Execute "Drop Table Curs"
```

הוספת שדה

תחביר משפט SQL להוספת שדה במסד הנתונים Access הוא:

```
Alter Table table_name Add Column column_name data_type
```

לדוגמה:

```
Alter Table Employees Add Column Pelephone Text (10)
```

גודל השדה ניתן להגדרה רק בשדות מסוג טקסט ובינארי. בכל משפט ניתן להוסיף שדה אחד בלבד ולכן, להוספת שדה יש לבצע קריאה נוספת לשירות Execute.

תחביר משפט SQL להוספת שדה ב- Oracle הוא:

```
Alter Table table_name Add(column_name data_type,
column_name data_type)
```

לדוגמה:

```
Alter Table Employees Add Pelephone VarChar(12)
```

שים לב שבמסד הנתונים Oracle **לא** ניתן למחוק שדה.

שינוי נתוני השדה

השינוי מתייחס לשינוי מאפייני השדה ולא לשינוי הערך שמכיל השדה. למשל, אפשר להפוך הגדרת שדה טקסט בגודל 20 תווים לשדה טקסט בגודל 30 וכדו'.

תחביר משפט SQL לשינוי נתוני שדה ב- Oracle הוא:

```
Alter Table table_name Modify(column_name data_type)
```

ניתן לשנות נתונים של כמה שדות במקביל, על ידי הפרדה ביניהם בפסיק. לדוגמה:

```
Alter Table Curs
Modify (Desc VarChar2(30), Manager VarChar2(25))
```

שינוי נתוני שדה מוגבל לתנאים הבאים:

1. מותר לשנות Null ל- Not Null (אפשר הפוך רק אם אין נתון בטבלה).

2. מותר לשנות מ-Num או Date ל- Char, להפך אסור.

3. מותר להרחיב גודל שדה, ולצמצם אסור.

שים לב ש**לא** ניתן לשנות נתוני שדה (הגדרה) במסד הנתונים Access.

הוספת אינדקס

כדי להוסיף אינדקס לטבלה קיימת במסד הנתונים, השתמש במשפט SQL הבא:

```
Create [unique] Index index_name
On table_name (Field1, Field2,... , Field13)
With Primary
```

אפשר לכתוב את המשפט בשורה אחת, אך לשם בהירות התחביר הוא נכתב כאן בשלוש שורות.

המילה "unique" שאחרי Create היא רשות, והיא מוספת כאשר האינדקס אינו מאפשר כפילויות. המשפט "With Primary" הוא גם כן רשות, הוא מוסף רק כאשר נרצה שהאינדקס יהיה **מפתח**. לדוגמה:

```
Create unique Index PrimaryKey
On Curs (CurId)
With Primary
```

דוגמה נוספת:

```
Create Index Names
On Employees (FirstName, LastName)
```

מחיקת אינדקס

למחיקת אינדקס מטבלה השתמש בתחביר הבא:

```
Drop Index index_name ON table_name
```

לדוגמה:

```
Drop Index Names ON Empolyees
```

סגירת מסד נתונים

בסיום עבודה עם מסד נתונים (סיום זה מתבצע לרוב כשהמשתמש יוצא מהיישום) יש לסגור אותו. סגירת מסד הנתונים מתבצעת בקריאה לשירות **Close** של המשתנה (מסוג Database) המייצג את מסד הנתונים. לדוגמה:

```
db.close
```

תרגיל לסיום הפרק

הנושאים שנלמדו בפרק זה רבים. כדי להטמיע אותם בצורה טובה נבצע את התרגיל הבא, המשלב את עיקרי הנושאים שנלמדו בפרק זה.

לצורך ביצוע התרגיל בנה את הטופס המוצג בתרשים זה:

תרשים 16.13

הטופס יאפשר למשתמש להקליד את קוד העובד ולקבל מידע אודותיו. ניתן יהיה למחוק את נתוני העובד מהטבלה, לעדכן אותה ולהוסיף עובד. לחיצה על הלחצן Exit תסגור את מסד הנתונים ותסיים את היישום.

את התרגיל בצע בשלבים:

1. צור מסד נתונים ושמור אותו בשם Work.mdb, בתיקיה שבה נמצאת ויז'ואל בייסיק.
2. הוסף למסד הנתונים Work.mdb טבלה בשם Employees.
3. הוסף לטבלה את השדות Id, Fname, Iname, Address, Tel.
4. צור אינדקס מפתח (PrimaryKey) לשדה Id.

5. עדכן את מאפייני הפקדים בטופס על פי הטבלה הזו:

פקד	מאפיין	ערך
FrmEmployees	Caption	תרגיל
lblId	Caption	Id:
lblFname	Caption	First Name:
lblLname	Caption	Last Name:
lblAddress	Caption	Address:
lblTel	Caption	Telephone:
txtId	Text	(Empty)
txtFname	Text	(Empty)
txtLname	Text	(Empty)
txtAddress	Text	(Empty)
txtTel	Text	(Empty)
cmdClear	Caption	Clear
cmdSearch	Caption	Search
cmdAdd	Caption	Add
cmdEdit	Caption	Edit
cmdDel	Caption	Delete
cmdExit	Caption	Exit

6. הצהר בחלק General Declaration שבטופס frmEmployees על המשתנים האלה:

```
Dim dbWork As Database
Dim recEmployees As Recordset
```

7. באירוע Form_Load קשר את מסד הנתונים ואת הטבלה.

```
Sub Form_Load ()
   Dim strFileName As String
   strFileName = App.Path & "\Work.mdb"
   Set dbWork = OpenDatabase(strFileName)
```

```
   Set recEmployees = dbWork.OpenRecordset("Employees")
End Sub
```

ברגע שהטופס FrmEmployees נטען, היישום מתקשר למסד הנתונים ולטבלה Employees.

8. בנה את השיגרה הבאה:

```
Private Sub Clear_Form()
     txtId.Text =""
     txtFname.Text =""
     txtLname.Text = ""
     txtAddress.Text = ""
     txtTel.Text = ""
     txtId.SetFocus
End Sub
```

מטרת השיגרה לנקות את הטופס מנתונים.

9. בנה את השיגרה הבאה:

```
Private Sub Search_Record(IdNum As Long)
   recEmployees.Index ="PrimaryKey"
   recEmployees.Seek "=", IdNum
   If recEmployees.NoMatch Then
      MsgBox ("There isn't Employee With This Id!")
   Else
      txtFname.Text = recEmployees![Fname]
      txtLname.Text = recEmployees![Lname]
      txtAddress.Text = recEmployees![Address]
      txtTel.Text = recEmployees![Tel]
   End If
End Sub
```

מטרת השיגרה לחפש רשומת עובד על פי מספר ת"ז שלו.

10. בנה את השיגרה הבאה:

```
Private Sub Add_Record()
   If Val(txtId.Text) = 0 Or Not IsNumeric(txtId.Text) Then
      MsgBox ("Please Enter Id Number!")
      txtId.Text =""
      txtId.SetFocus
   Else
      recEmployees.AddNew
      recEmployees![Id] = Val(txtId.Text)
      recEmployees![Fname] = txtFname.Text
      recEmployees![Lname] = txtLname.Text
      recEmployees![Address] = txtAddress.Text
      recEmployees![Tel] = txtTel.Text
      recEmployees.Update
   End If
End Sub
```

מטרת השיגרה להוסיף עובד חדש.

11. בנה את השיגרה הבאה:

```
Private Sub Edit_Record()
   recEmployees.Edit
   recEmployees![Fname] = txtFname.Text
   recEmployees![Fname] = txtFname.Text
   recEmployees![Lname] = txtLname.Text
   recEmployees![Address] = txtAddress.Text
   recEmployees![Tel] = txtTel.Text
   recEmployees.Update
End Sub
```

מטרת השיגרה לערוך נתוני עובד קיים.

12. בנה את השיגרה הבאה:

```
Private Sub Move_Next_Record()
   If recEmployees.EOF Then
      recEmployees.MoveLast
   Else
      recEmployees.MoveNext
   End If
   txtId.Text = recEmployees![ID]
   txtFname.Text = recEmployees![Fname]
   txtLname.Text = recEmployees![Lname]
   txtAddress.Text = recEmployees![Address]
   txtTel.Text = recEmployees![Tel]
End Sub
```

מטרת השיגרה להציג את נתוני העובד הבא.

13. כתוב את שורות הקוד הבאות:

```
Private Sub cmdSearch_Click()
   Search_Record Val(txtId.Text)
End Sub

Private Sub cmdClear_Click()
   Clear_Form
End Sub

Private Sub cmdAdd_Click()
   Add_Record
End Sub

Private Sub cmdEdit_Click()
   Edit_Record
End Sub
```

```
Private Sub cmdDel_Click()
   recEmployees.Delete
   Move_Next_Record
End Sub

Private Sub cmdExit_Click()
   dbWork.Close
   Unload Me
End Sub
```

14. הקש F5 והפעל את היישום.

תרגול

1. הוסף לטופס שבתרגיל את הלחצנים "Next" ו-"Previous". לחיצה על Next תציג את הרשומה הבאה, לחיצה אל Previous תציג את הרשומה הקודמת. שים לב למקרי גבול, שבהם המצביע נמצא ברשומה הראשונה או האחרונה.
2. הוסף לקוד התרגיל גם טיפול במקרה בו ישנה רשומה אחת בלבד והמשתמש לוחץ על לחצן Delete.
3. הוסף לחצן Query אשר יציג את תוצאת השאילתה הבאה: שמם המלא ואת מספר הטלפון של העובדים אשר קוד העובד שלהם גדול מ- 100.

17 :
הוספת סרגל כלים ליישום

ברוב היישומים בסביבה חלונאית קיים סרגל כלים שנמצא בדרך כלל תחת שורת התפריטים. תרשים 17.1 מציג את סרגל הכלים הסטנדרטי של ויז'ואל בייסיק.



תרשים 17.1

סרגל הכלים מאפשר ביצוע פעולות שכיחות ביישום על ידי גישה מהירה אליהן. סרגל הכלים אינו תחליף לתפריטים או לחלק מהם, אלא כתוספת. את כל הפעולות המבוצעות באמצעות סרגל הכלים ניתן לבצע משורת התפריטים.

סרגל הכלים יכלול בדרך כלל את הפעולות השכיחות שהמשתמש מבצע באופן תדיר, כשמירה ופתיחה של קובץ ועוד. כדי להקל על המשתמש את השימוש החוזר ונשנה בשורת התפריטים לביצוע פעולות שכיחות אלו, אנו יוצרים לו לחצני **קיצור דרך** בצורת סרגל הכלים. במקום לעבור את המסלול **קובץ**, **שמור** (בשורת התפריטים) בכל פעם שהמשתמש ירצה לשמור את הקובץ בזיכרון, הוא ילחץ על הלחצן  בסרגל הכלים.

כשהצגנו את הפקד **CommandButton** הזכרנו את היכולת ליצור סרגל כלים מאולתר בעזרת כמה פקדים כאלה. בפרק זה נלמד לבנות סרגל כלים מקצועי באמצעות הפקד המיועד לכך.

הפקד ליצירת סרגל כלים הינו **ToolBar** .

הפקד ImageList

הפקד **ImageList**  מאחסן רשימת תמונות המהוות את מאגר התמונות של לחצני סרגל הכלים. שים לב שברגע שתמונה אוחסנה בפקד, ויז'ואל בייסיק אינה זקוקה לשם קובץ התמונה ולנתיב. גם כאשר היישום יפעל במחשב אחר שאין בו את אותם קבצי תמונות האגורים בפקד ImageList, היישום יפעל ללא תקלות. התמונות תילקחנה מהפקד ImageList עצמו.

כדי להוסיף את הפקד לארגז הכלים יש לבצע את הפעולות הבאות:

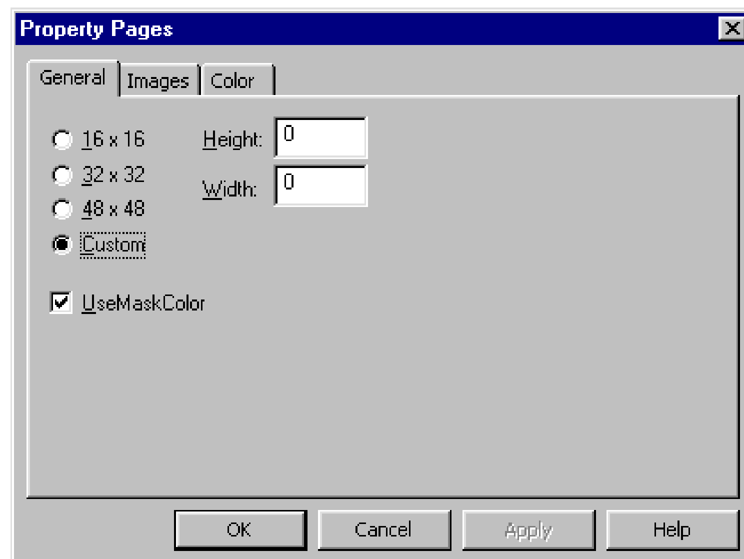
1. לחץ לחיצה ימנית על ארגז הכלים.
2. בחר באפשרות **Components**. בשלב זה יופיע החלון Components המציג את רשימת האובייקטים שתוכל להוסיף כפקדים לארגז הכלים.
3. סמן את האובייקט Microsoft Windows Common Controls 6.0.
4. אשר את הבחירה.

לאחר אישור הפעולה, נוספה לארגז הכלים קבוצת פקדים הכוללת בתוכה את הפקד ImageList. בחר בפקד **ImageList** והצב אותו על הטופס. מיקומו בטופס אינו חשוב, מכיון שפקד זה לא נראה לעיני המשתמש.

בשלב זה כל שנותר הוא לקבוע מאפיינים לפקד ולבנות את מאגר התמונות.

קביעת גודל התמונה

כדי לקבוע את מאפייני הפקד, לחץ לחיצה ימנית על הפקד (הממוקם בטופס) ובחר באפשרות **Properties**. בחירה זו גורמת להצגת החלון **Property Pages**, ובו שלוש כרטיסיות (תרשים 17.2).



תרשים 17.2

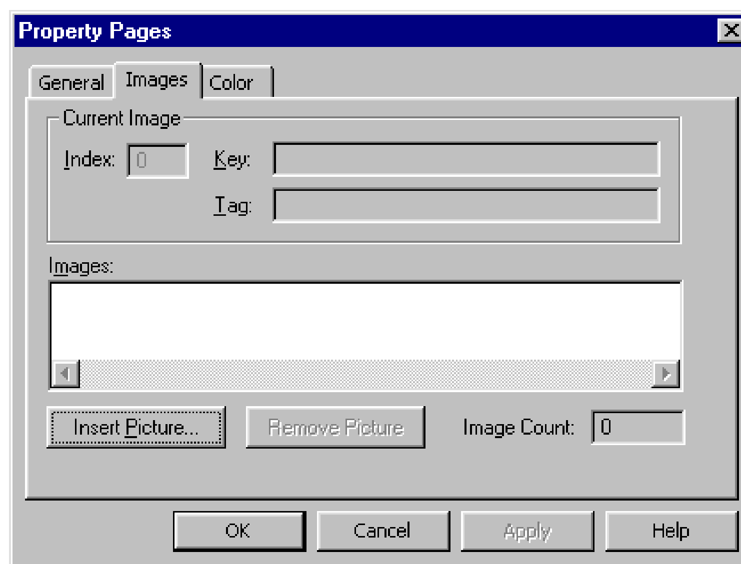
בכרטיסיה **General** נקבע גודל התמונות שבמאגר הפקד. שלוש האפשרויות הראשונות מאפשרות בחירה של גודל מובנה. האפשרות הרביעית Custom מאפשרת

קביעה ידנית של גודל התמונות. במקרה זה, נקבע בתיבות הטקסט Height ו- Width את גובה ורוחב התמונות בהתאמה.

שים לב, שבחירת גודל תמונה קובעת את הגודל הנבחר **לכל** התמונות. ברגע שקיימות תמונות במאגר **לא** ניתן לשנות את גודלן.

בחירת תמונה

הכרטיסיה **Images** מאפשרת לבנות את מאגר התמונות (תרשים 17.3).



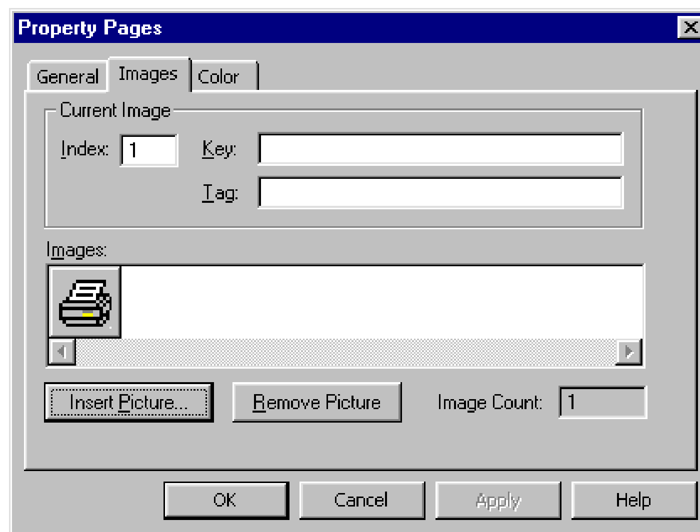
תרשים 17.3

כדי להוסיף תמונה למאגר, יש לבצע את הפעולות האלו :

1. לחץ על לחצן **Insert Picture**. לחצן זה פותח חלון **Select Picture**, המאפשר חיפוש ובחירת קובץ תמונה מהזיכרון.
2. בחר קובץ תמונה ואשר את הבחירה (לחצן Open).

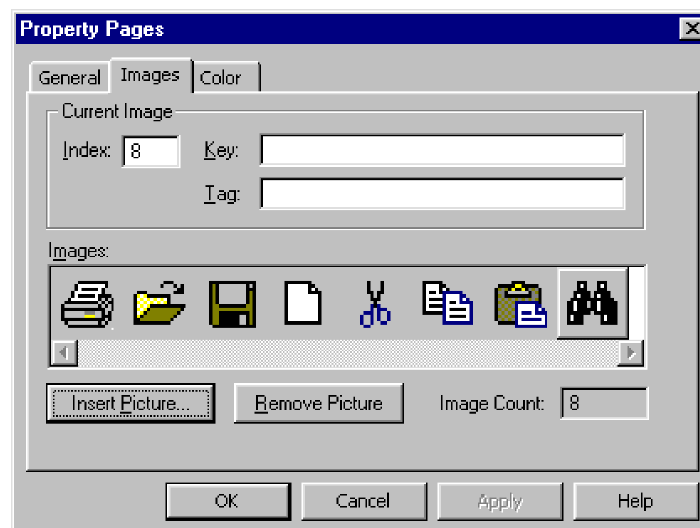
בשלב זה נוספה תמונה למאגר וקיבלה ערך Index = 1. אינדקס 1 מציין כי זו התמונה הראשונה במאגר התמונות. מלבד זה, ערך האינדקס של התמונה הוא הערך שנציין בכל פעם שנרצה להתייחס לתמונה (תרשים 17.4).

חזור על הפעולות 1 ו- 2 כדי להוסיף תמונות למאגר (תרשים 17.5). לחילופין, בחר בתמונה כלשהי בלחיצה אחת של העכבר עליה ומחק אותה מהמאגר בלחיצה על הלחצן Remove Picture.



תרשים 17.4

כל תמונה חדשה נוספת למאגר בסופו, אך ניתן להוסיף תמונה למקום כלשהו במאגר. לצורך זה נבחר בתמונה במאגר שלימינה רוצים להוסיף תמונה ונבצע את שני השלבים להוספת תמונה. במקרה זה התמונה תתוסף לימין התמונה שנבחרה, ולא בסוף המאגר. ערך האינדקס של התמונה שנוספה ושל אלו שאחריה ישתנה בהתאם.

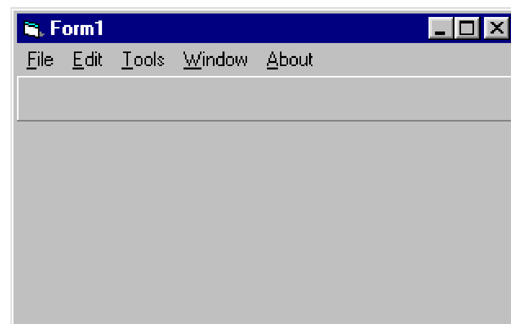


תרשים 17.5

הפקד ToolBar

פקד נוסף בקבוצת הפקדים שנוספה לארגז הכלים מבחירת האובייקט Microsoft Windows Common Controls 6.0 הוא הפקד **ToolBar** . בעזרתו נבנה את סרגל הכלים שלנו.

נבחר בפקד זה מארגז הכלים ונציב אותו בטופס. נוכל להבחין בכך שהפקד "קופץ" אוטומטית לראש הטופס תחת שורת הכותרת, למקום שבו נמצא בדרך כלל סרגל הכלים של היישום. הסרגל מוצג בצורה "שטוחה", אך ניתן לקבוע לו מראה תלת-מימדי על ידי שינוי ערך המאפיין BorderStyle שלו ל- 1 (תרשים 17.6).



תרשים 17.6

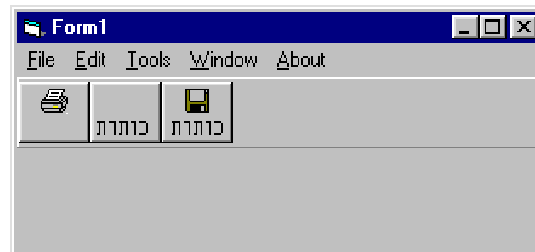
לאחר הצבת הפקד ToolBar בטופס, כל שנותר לעשות הוא לעדכן את המאפיינים שלו ולבנות את סרגל הכלים באופן הרצוי. השלב הראשון בבניית סרגל הכלים הוא קישורו לפקד ImageList.

קישור ToolBar ל- ImageList

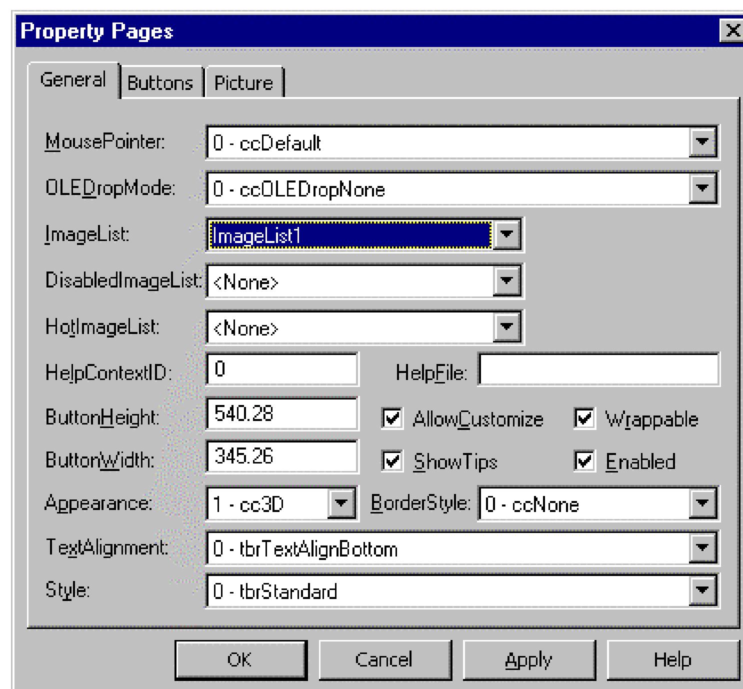
הקשר בין שני הפקדים בא לידי ביטוי בכך שהפקד ImageList הוא מאגר התמונות לחצני סרגל הכלים. שים לב שאין חובה להציג בסרגל הכלים דווקא תמונות, ואפשר שהם יופיעו עם כיתוב עליהם. מקובל ביישומים חלונאיים שסרגל הכלים מכיל לחצנים עם תמונות. ניתן לשלב בין השניים, כלומר, לחצן הכולל תמונה וגם כיתוב תחתיה (תרשים 17.7).

כדי לבצע את הקשר בין שני הפקדים, לחץ לחיצה ימנית על הפקד **ToolBar** שנמצא בטופס ובחר באפשרות **Properties**. בחירה זו מציגה חלון **Property Pages** בעל שלוש כרטיסיות: General, Buttons ו- Picture. בכרטיסיה General מופיעה קבוצה של תיבות נפתחות, השנייה שבהם היא ImageList. תיבה נפתחת זו מכילה רשימה של כל הפקדים מסוג ImageList (אם קיימים כמה פקדים כאלה בטופס, כולם יוצגו ברשימה). מתוך רשימה זו נבחר בשם הפקד המייצג את ImageList שאליו רוצים

לקשר את סרגל הכלים האישי. מעתה סרגל הכלים הזה יתייחס לפקד זה ולמאגר התמונות שבו בלבד (תרשים 17.8).



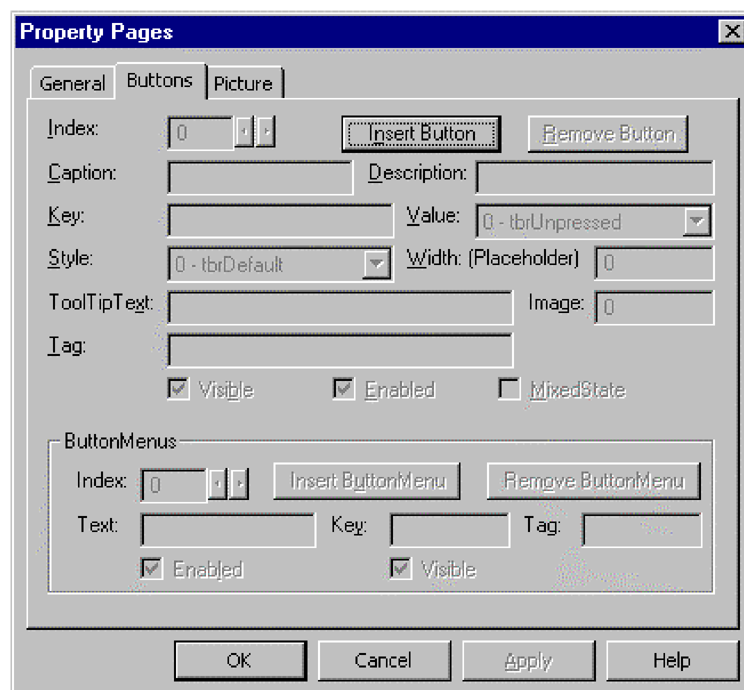
תרשים 17.7



תרשים 17.8

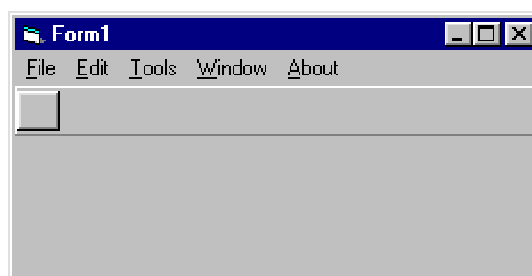
הוספת לחצנים לסרגל

לאחר שנוצר הקשר בין סרגל הכלים למאגר התמונות, נבנה את סרגל הכלים ונוסיף לו את הלחצנים המתאימים. לצורך כך נעבור לכרטיסיית **Buttons** (תרשים 17.9).



תרשים 17.9

לחיצה על הלחצן **Insert Button** גורמת להוספת לחצן ריק לסרגל. גודל הלחצן יהיה כגודל אשר נקבע בפקד Image List לגודל תמונות. הלחצן הנוסף מקבל אוטומטית את ערך האינדקס 1, המורה שזהו הלחצן הראשון בסרגל. לחצן זה יופיע בצד השמאלי ביותר של סרגל הכלים וכל לחצן שיתוסף יצורף לימין הלחצן הקודם לו (תרשים 17.10).



תרשים 17.10

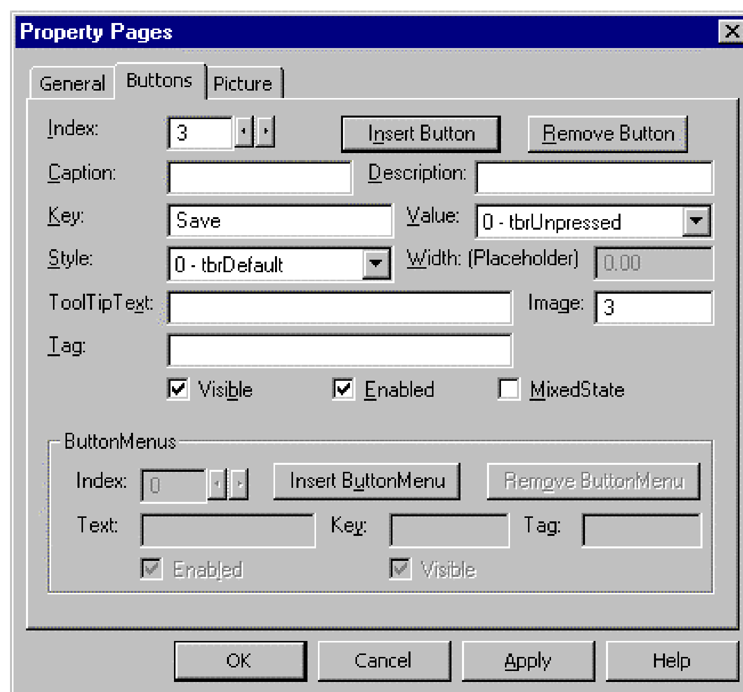
ניתן להכניס לחצן בין שני לחצנים קיימים: עדכן את תיבת הטקסט **Index** שבכרטיסיה **Buttons** לערך האינדקס של הלחצן שלימינו ברצונך להוסיף אותו ולחץ על **Insert Button**. הלחצן יתוסף בסרגל במקום המתאים לערך האינדקס שלו.

כדי לבטל לחצן ולהסירו מסרגל הכלים, עדכן את תיבת הטקסט Index בערך האינדקס של הלחצן ולחץ על הלחצן **Remove Button**.

קביעת מאפיינים ללחצן

לאחר הוספת הלחצנים לסרגל צריך לקבוע לכל לחצן את המאפיינים. המאפיינים מעודכנים בתיבות הטקסט השונות שבכרטיסיה **Buttons**. המאפיינים העיקריים הם:

- **Caption** - כותרת הלחצן. בנוסף לתמונת הלחצן ניתן להוסיף תחתיה כיתוב (או כותרת).
- **Key** - מפתח הלחצן. מפתח זה הוא מחרוזת המזהה את הלחצן בין שאר הלחצנים. בכל פעם שנפנה ללחצן מסוים נציין את המפתח שלו (תרשים 17.11). מפתח הלחצן ממלא תפקיד זהה למאפיין Name של פקד רגיל.
- **ToolTipText** - מאפיין זה קובע מה יוצג בתיבה המופיעה באופן אוטומטי כאשר סמן העכבר נמצא על גבי הלחצן, וללא לחיצה עליו.
- **Image** - מספר התמונה. לכל תמונה במאגר התמונות שבפקד Image List יש מספר אינדקס. במספר התמונה קובעים את השיוך של כל לחצן לתמונה שלו, על ידי ציון ערך האינדקס של התמונה במאפיין Image של הלחצן.



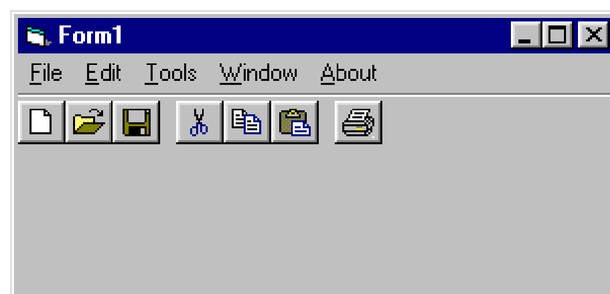
תרשים 17.11

לחצני עזר נוספים

כשיוצרים לחצן בעזרת Insert Button נוצר לחצן סטנדרטי, או במילים אחרות, במאפיין Style ערך הלחצן הוא brDefault - 0. כל לחצן נוסף נוצר בסגנון דומה לקודמו, וכך נוצרת שורה שלמה של לחצנים המרכיבים את הסרגל כולו.

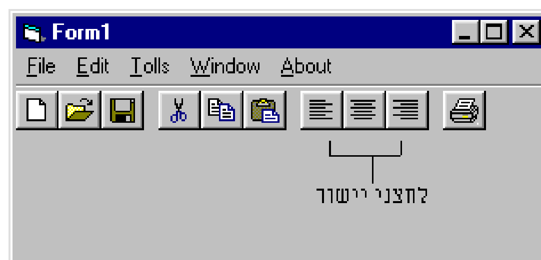
לפעמים נרצה לקבץ לחצנים בקבוצות **לוגיות** שונות, כמו לדוגמה קבוצת לחצנים המטפלת בקובץ (חדש, שמירה, פתיחה וכו'). במקרה זה, נרצה להפריד את הקבוצות הפרדה פיזית בהתאם להפרדה הלוגית.

למטרה זו מספקת לנו ויז'ואל בייסיק סוג נוסף של לחצן, **לחצן מפריד**, אשר שונה מלחצן רגיל בערך המאפיין Style שלו. כדי לקבוע שלחצן כלשהו הוא לחצן מפריד, נבחר מתוך הרשימה הנפתחת Style בערך **tbrSeparator - 3**. בכל פעם שנרצה להפריד בין קבוצת לחצנים, ניצור ביניהם לחצן מפריד (תרשים 17.12). בלחצן מפריד אין משמעות למאפיינים שצוינו בסעיף הקודם ואין צורך לעדכן אותם, מלבד המאפיין Style.



תרשים 17.12

ערך נוסף שמקבל המאפיין Style הוא **tbrButtonGroup - 2**. בערך זה נבחר כאשר נרצה לאחד באופן **פיזי** קבוצת לחצנים. משמעות קיבוץ הלחצנים באופן זה היא, שניתן לבחור רק בלחצן אחד בלבד מתוך הקבוצה, בדומה לפקד OptionButton. דוגמה לשימוש בסגנון זה הם לחצני היישור במעבדי התמלילים (תרשים 17.13).



תרשים 17.13

ערך נוסף שמקבל המאפיין Style הוא **tbrCheck - 1**. ערך זה מתפקד כפקד CheckBox בסגנון הגרפי שלו. כאשר הלחצן נבחר הוא שקוע ובחירה נוספת בו תשחזרו למצבו הרגיל (תרשים 17.14).



תרשים 17.14

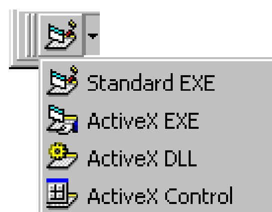
ערך נוסף שמקבל המאפיין Style הוא **Placeholder - 4**. סגנון זה יוצר שטח פנוי על סרגל הכלים כדי לאפשר לנו להציב פקד שיהיה חלק מהסרגל. לדוגמה, התיבה הנפתחת ComboBox במעבד התמלילים Microsoft Word המכילה את שמות הגופנים הקיימים במערכת (תרשים 17.15).



תרשים 17.15

כאשר בוחרים בסגנון Placeholder, תיבת הטקסט :Width הופכת באופן אוטומטי לפעילה. בתיבה זו נקבע רוחב השטח על סרגל הכלים אשר יהיה פנוי לטובת הפקד.

ערך חדש לגירסה 6 אותו מקבל המאפיין Style הוא Dropdown - 5. סגנון זה יוצר לחצן נפתח המכיל תחתיו כמה לחצנים. דוגמה לכך הוא הלחצן **Add Project** (תרשים 17.16).



תרשים 17.16

בקרת הלחצן משורות הקוד

לאחר שיוצרים את סרגל הכלים, יש לכתוב את שורות הקוד אשר יבוצעו בלחיצת המשתמש על אחד הלחצנים שבסרגל. בדומה לפקד רגיל, המוכוון על ידי אירועים, גם לסרגל הכלים אירועים שונים שהוא יודע להגיב עליהם. האירוע העיקרי הוא **ButtonClick**, שמתרחש כאשר המשתמש לוחץ על אחד הלחצנים בסרגל.

כדי להציג "שלד" של שיגרת לחצן, צריך ללחוץ על הסרגל לחיצה כפולה במבט עיצוב. לדוגמה, שורות הקוד הבאות יופיעו בחלון הקוד עבור סרגל הכלים tbrStandard :

```
Private Sub tbrStandard_ButtonClick(ByVal Button As _
            ComctlLib.Button)
End Sub
```

השיגרה מקבלת כפרמטר את המשתנה Button המייצג את לחצני הסרגל. כל התייחסות למאפייני אחד הלחצנים תיעשה באמצעות משתנה זה.

כזכור, פנייה לאחד הלחצנים בסרגל נעשית בציון ערך המאפיין **Key** שלו. על כן, כדי לברר על איזה לחצן לחץ המשתמש, נברר את ערך המאפיין Key של המשתנה Button המייצג את הלחצן שנלחץ. יוצא, כי שורות הקוד שיבואו בתוך השיגרה ButtonClick, יכללו מבנה קבלת החלטה המברר את ערך Button.Key.

לפניך מודגמות מספר שורות קוד. הוסף אותן לשורות הקוד בתוכנית שלך :

```
Private Sub tbrStandard_ButtonClick(ByVal Button As _
                              ComctlLib.Button)
    Select Case Button.Key
        Case "Open"  ' Open file.
            ' Add code to Open a file
        Case "Save"  ' Save file.
            ' Add code to Save a file
        Case Else
            ' If any other button is pressed
    End Select
End Sub
```

תרגול

1. בנה סרגל כלים (מפקדי CommandButton) המכיל את האפשרויות פתיחת קובץ קיים, יצירת קובץ חדש ושמירת קובץ.
2. בנה את אותו סרגל, אך הפעם עשה זאת על ידי הפקד המתאים.

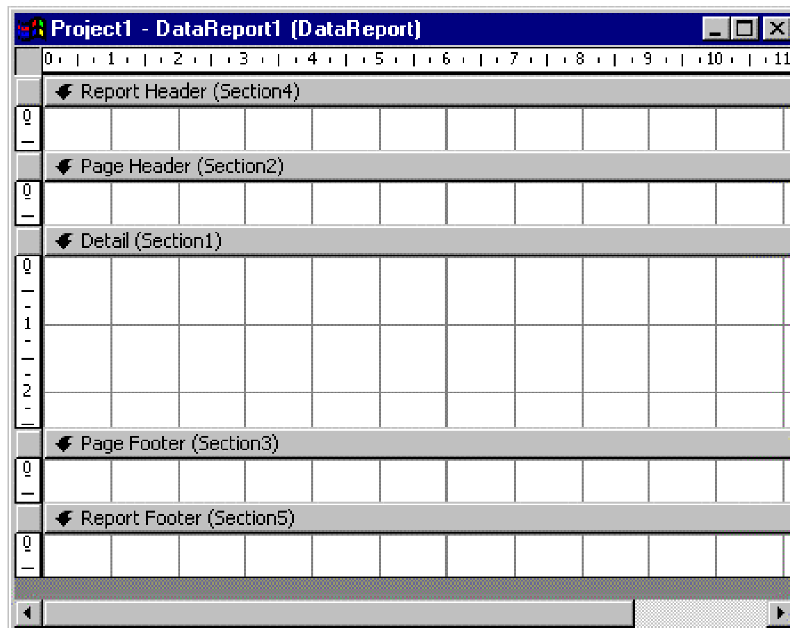
18: דוחות

חלק בלתי נפרד וגם חשוב ביישום הוא ה**דיווח**. מטרת היישום לספק למשתמש כלי בעל ממשק נוח וידידותי לשמירת הנתונים שלו, אשר נשמרים במסד הנתונים. אולם בזה לא תם התפקיד, כי המשתמש רוצה לצפות בנתונים ששמר ובמקרים רבים הוא רוצה לראות חלקים שונים בחתכים מוגדרים, בצורה של חישובים סטטיסטיים, גרפיים, על פי מיון ועוד. כאן נכנסת לתמונה יכולת המתכנת בבניית דוחות מתאימים, אשר יציגו למשתמש את המידע המבוקש.

יצירת דוח

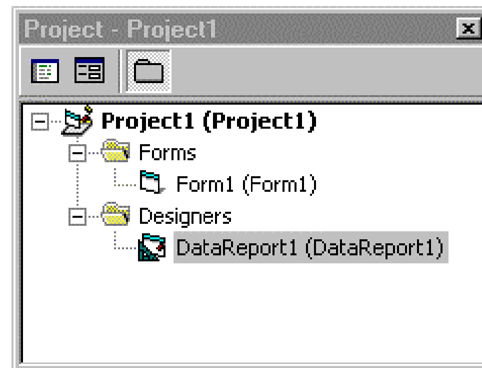
בגרסאות קודמות של ויז'ואל בייסיק לא ניתן היה ליצור דוח, אך כעת הדבר אפשרי. יצירת דוח נעשתה בעזרת מחולל הדוחות **Crystal Reports 4.6** של חברת Seagate Software שמצורף למערכת ויז'ואל בייסיק. כיום יש גירסה מתקדמת יותר של התוכנה בשם **Crystal Reports 6**, שאפשר לרכוש בנפרד מתוכנת ויז'ואל בייסיק. בגירסה **6** נוסף רכיב **DataReport** המאפשר ליצור דוח פשוט כחלק מהפרויקט.

כדי להוסיף את DataReport לפרויקט נבחר באפשרות **Add Data Report** שבתפריט Project. בשלב זה נוסף לפרויקט טופס גלמי לעיצוב דוח (תרשים 18.1).



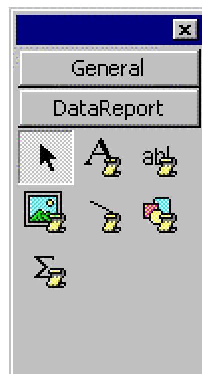
תרשים 18.1

DataReport הוא קובץ נוסף בפרויקט, בעל סיומת **Dsr**. שים לב שהוא מופיע בחלון הפרויקט ככל קובץ אחר (תרשים 18.2).



תרשים 18.2

במקביל מספקת תוכנת ויז'ואל בייסיק מספר פקדים לשימוש בדוח, כמו למשל, הוספת כיתוב, תמונה ועוד (תרשים 18.3). הפקדים בארגז הכלים מוגדרים תחת הכותרת DataReport.

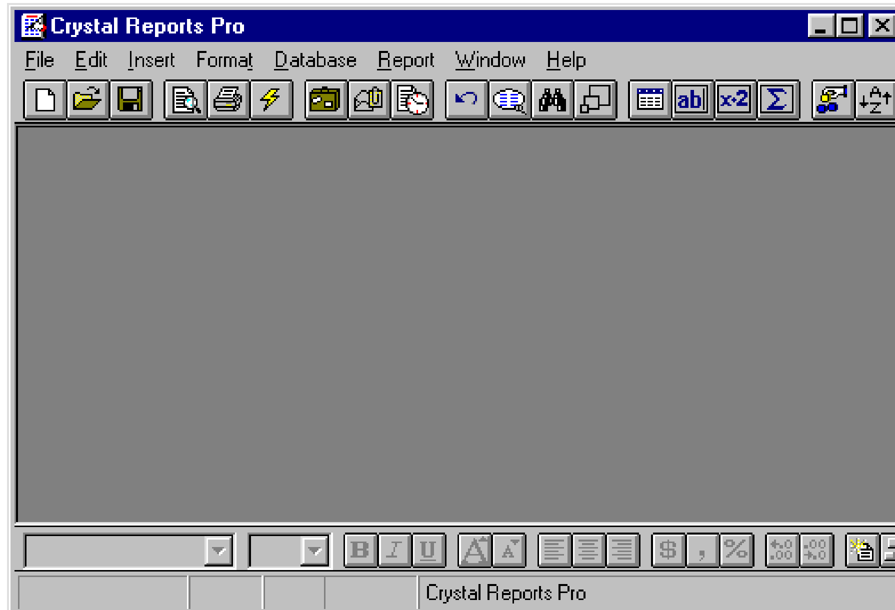


תרשים 18.3

למרות שניתן ליצור בוויז'ואל בייסיק (בגירסה 6 בלבד) דוח כחלק מהפרויקט, דוח זה פשוט ומוגבל. כלי טוב יותר שנעשה בו שימוש בגרסאות הקודמות ועדיין ניתן להשתמש בו בגירסה 6 הוא התוכנה **Crystal Report**. כלי זה יוצר דוח בקובץ בלתי תלוי ועצמאי (בעל סיומת **rpt**), שניתן לנהל אותו מתוך הפרויקט בוויז'ואל בייסיק. מכיון שהאפשרויות לבניית דוח בכלי זה רבות ומגוונות יותר, נתמקד בו בפרק זה.

סוגי דוחות

בחירה באפשרות **Crystal Reports** פותחת את **מחולל הדוחות** (תרשים 18.4).



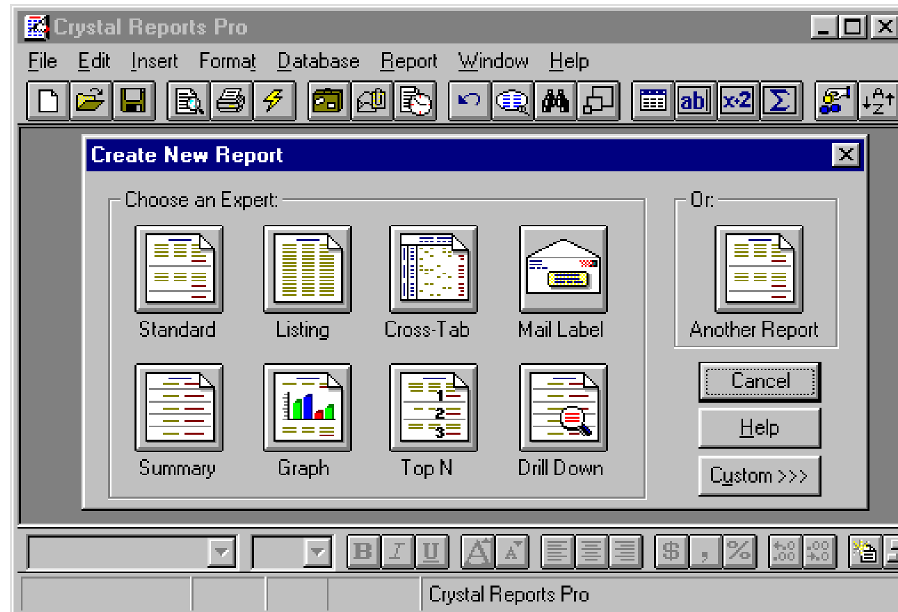
תרשים 18.4

בחר באפשרות **New** שבתפריט **File** כדי ליצור דוח חדש. בשלב זה נפתח החלון **Create New Report** בו מוצגים סגנונות שונים של דוחות (תרשים 18.5) שניתן ליצור בעזרת מחולל הדוחות. הסגנונות שמספק מחולל הדוחות הם:

- **Standard** - ליצירת דוח מפורט עם סיכומי ביניים (וסופיים).
- **Listing** - ליצירת דוח מפורט ללא סיכומי ביניים.
- **Cross-Tab** - ליצירת דוח טבלאי.
- **Mail Label** - ליצירת דוח בפורמט תוויות דואר.
- **Summary** - ליצירת דוחות סיכומיים בלבד (ללא פירוט).
- **Graph** - ליצירת דוח גרפי.
- **Top N** - ליצירת דוח הכולל מידע על הנתונים ה"גדולים ביותר" (כדוגמת עשרת המקומות הראשונים).
- **Drill Down** - ליצירת דוח לפי קטגוריות.

בנוסף, המתכנת יכול ליצור דוח כרצונו, שאינו אחד מהסגנונות המוצעים על ידי מחולל הדוחות. ליצירת דוח ידני (בניגוד לדוח סטנדרטי) לחץ על הלחצן **Custom**. גם

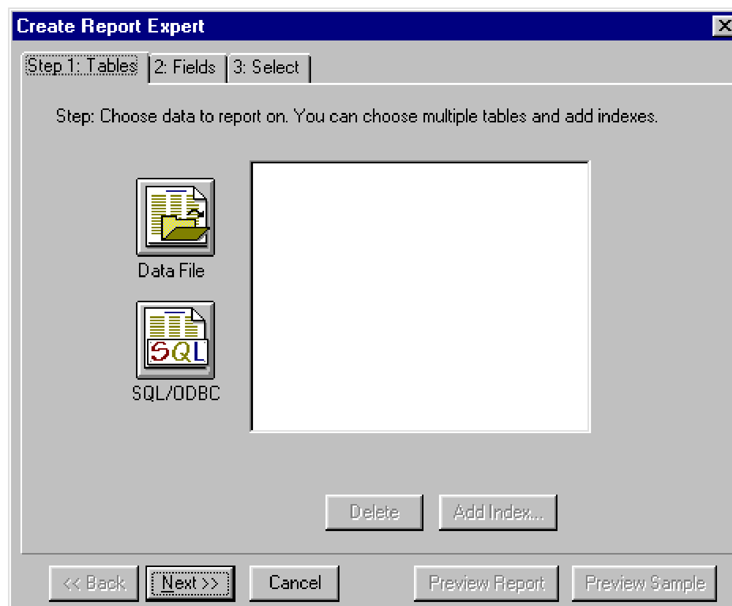
אם בוחרים בדוח סטנדרטי, אין כל קושי לשנות את העיצוב שלו באופן ידני ללא עזרת האשף (wizard) של מחולל הדוחות.



תרשים 18.5

שלבים בבניית דוח

לאחר בחירה בסגנון המתאים לדוח (ניקח לדוגמה את הסגנון הפשוט יותר - **Listing**, או דיווח רשימה, מחולל הדוחות מציג **אשף דוחות** (תרשים 18.6), אשר בהנחיה שלנו יוצר את הדוח. לאחר בחירת סגנון Listing יופיע החלון **Create Report Expert**, אשר מכיל כרטיסיות שונות הכוללות את כל שלבי יצירת הדוח, על פי סוג הדוח שנבחר.



תרשים 18.6

בחירת מסד הנתונים

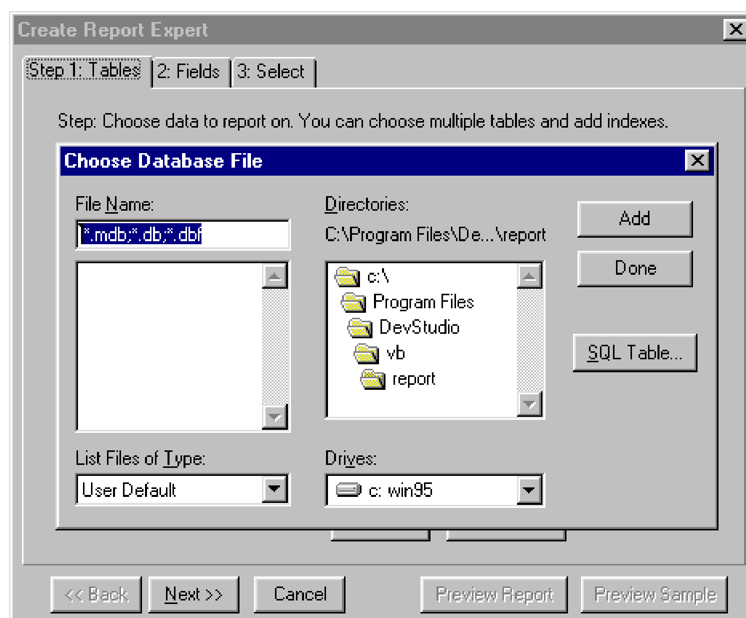
השלב הראשון ביצירת דוח הוא **קישורו** למסד הנתונים. בעזרת קשר זה דולה מחולל הדוחות את הנתונים ממסד הנתונים, כפי שנגדיר לו בהמשך, ומציג אותם בדוח. לחץ על הלחצן **Data File** שבכרטיסיה **Tables**.

הלחצן Data File מקשר אותנו לקובץ מסד נתונים כדוגמת mdb, db ו- dbf. לקישור דרך ODBC למסד נתונים הנמצא בשרת, כפי שדרוש לקישור אל מסד נתונים של Oracle לדוגמה, לחץ על הלחצן SQL/ODBC.

לחיצה על לחצן Data File פותחת את החלון **Choose Database File** (תרשים 18.7).

מחלון זה נבחר את מסד הנתונים שאליו נרצה לקשר את הדוח. לאחר בחירה בקובץ המתאים, נלחץ על הלחצן **Add** כדי להוסיף את הקובץ למחולל הדוחות. לחיצה על הלחצן Add אינה סוגרת עדיין את החלון Choose Database File. הסיבה לכך נעוצה בעובדה שניתן להוסיף קובץ נוסף ובמילים אחרות, הדוח יכול להכיל נתונים ממספר מסדי נתונים.

לאחר בחירת הקובץ (או הקבצים) המתאים יש ללחוץ על הלחצן **Done** כדי לחזור לאשף הדוחות.



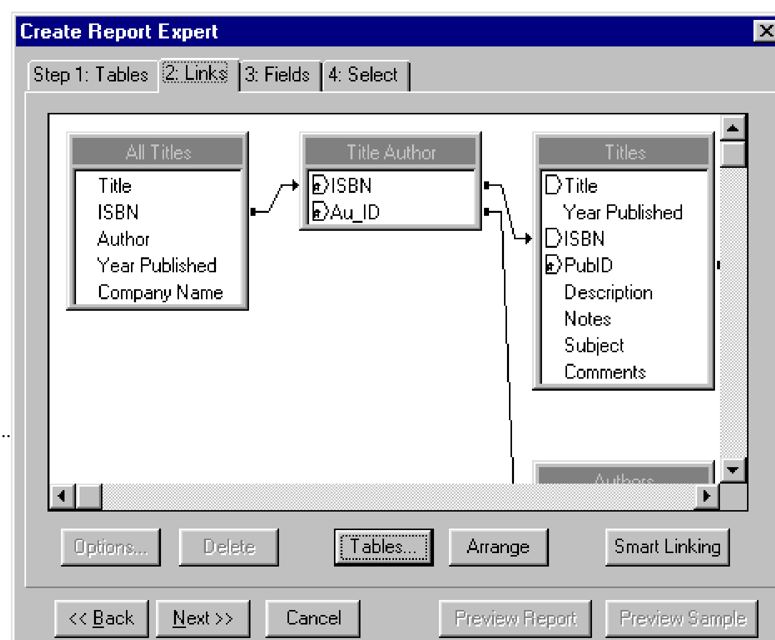
תרשים 18.7

קשרים בין טבלאות

אם מסד הנתונים כולל מספר טבלאות המקיימות ביניהן קשרי גומלין, תתוסף לחלון Create Report Expert הכרטיסייה **Links** המציגה את הטבלאות ואת הקשרים שביניהן. בכרטיסייה זו ניתן לשנות או למחוק קשרים קיימים.

יצירת קשר חדש

כדי ליצור קשר חדש בין טבלאות, בחר בשדה המקשר מאחת הטבלאות, גרור אותו בעזרת העכבר והצב אותו על השדה המקשר בטבלה השנייה. בסיום הפעולה, מחולל הדוחות מותח באופן אוטומטי קו בין שתי הטבלאות ומקשר אותן באמצעות השדה המקשר (תרשים 18.8).



תרשים 18.8

לאחר שנוצר הקשר בין הטבלאות, נלחץ על **Options** כדי להגדיר את סוג הקשר (תרשים 18.9).

מחיקת קשר

כדי לבטל קשר גומלין בין שתי טבלאות, בחר בקשר (לחץ לחיצה אחת על הקו שנמתח בין שתי הטבלאות) ולחץ על **Delete** (או הקש Delete).

שים לב, ניתן תמיד לחזור שלב אחד בלחיצה על הלחצן **Back** או בבחירת הכרטיסיה הרצויה. לדוגמה, אם ברצונך למחוק מספר טבלאות מהדוח, חזור לכרטיסיה **Tables**, מרשימת הטבלאות בחר בטבלה אותה אתה מעוניין למחוק ולחץ על **Delete**.

לאחר שסיימת את פעולת הקישור לחץ על Next כדי לעבור לשלב הבא.

בחירת השדות שיוצגו בדוח

השלב הבא מוצג בכרטיסיה **Fields** (תרשים 18.10).

Link Options

From Table: All Titles
To Table: Title Author

ISBN --> ISBN

Index In Use:
<no specific index>

Fields In Index:

Allow partial text matches

When linking to two files from this file:
Look up both at the same time.
Look up all of one, then all of others.
Look up all the combinations of the two files.

SQL Join Type:
Equal [=]
Left Outer [=(+), *=]
RightOuter [(+)=, =*]
Greater [>]
Less [<]
Greater Or Equal [>=]
Less Or Equal [<=]
Not Equal [!=]

OK Cancel Help

תרשים 18.9

Create Report Expert

Step 1: Tables | 2: Links | 3: Fields | 4: Select

Step: Select fields to include in report. You can reorder them and change headings.

Database Fields:
Report Fields:
Database Fields:
-----Authors-----
Au_ID
Author
Year Born
-----Publishers-----

Add ->
All ->>
<- Remove
<<-All

Report Fields:

Browse Data... Formula... Column Heading:

<< Back Next >> Cancel Preview Report Preview Sample

תרשים 18.10

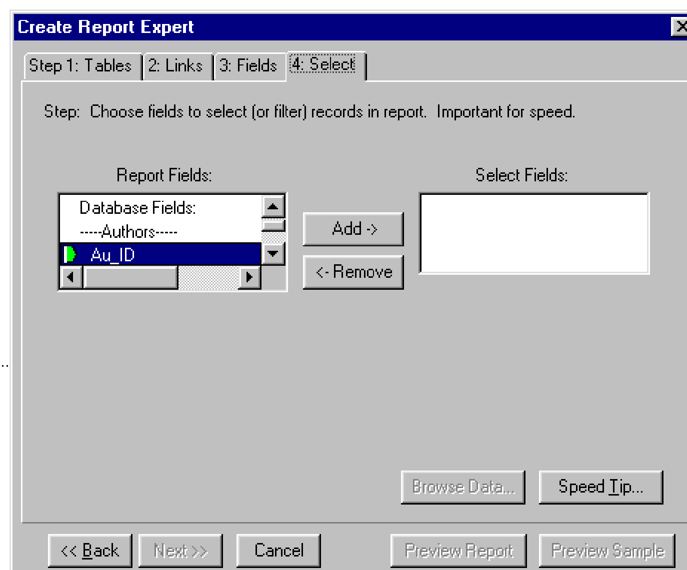
בשלב זה נורה לאשף איזה שדות מתוך אילו טבלאות במסד הנתונים ברצוננו להציג בדוח. אין צורך לכלול את כל הטבלאות הקיימות במסד הנתונים, וגם לא את כל השדות שכלולים בהן. ניתן להציג חלק מהשדות שבטבלה אחת ואף לא שדה אחד מטבלה אחרת. כדי לבחור איזה שדה יוצג בדוח בצע את הפעולות הבאות:

1. בחר בשדה הרצוי מתוך תיבת הרשימה **Database Fields** (הרשימה השמאלית).
2. לחץ על הלחצן **Add** כדי להעביר את השדה לתיבת הרשימה **Report Fields**.
3. בתיבת הטקסט **Column Heading** רשום את הכותרת לשדה שתופיע בדוח. כאן ניתן להחליף את שמות השדה הכתובים באנגלית לכותרות בעברית.

- חזור על פעולות אלו על כל שדה שאתה רוצה להציג בדוח.
- כדי להוסיף את **כל** השדות לדוח, לחץ על **All** וכל השדות יועתקו לתיבת הרשימה Report Fields.
- לביטול הצגת שדה, בחר בו מתיבת הרשימה Report Fields ולחץ על **Remove**.
- לאחר שסיימת לבחור בשדות שיוצגו בדוח לחץ על **Next** ועבור לשלב הבא.

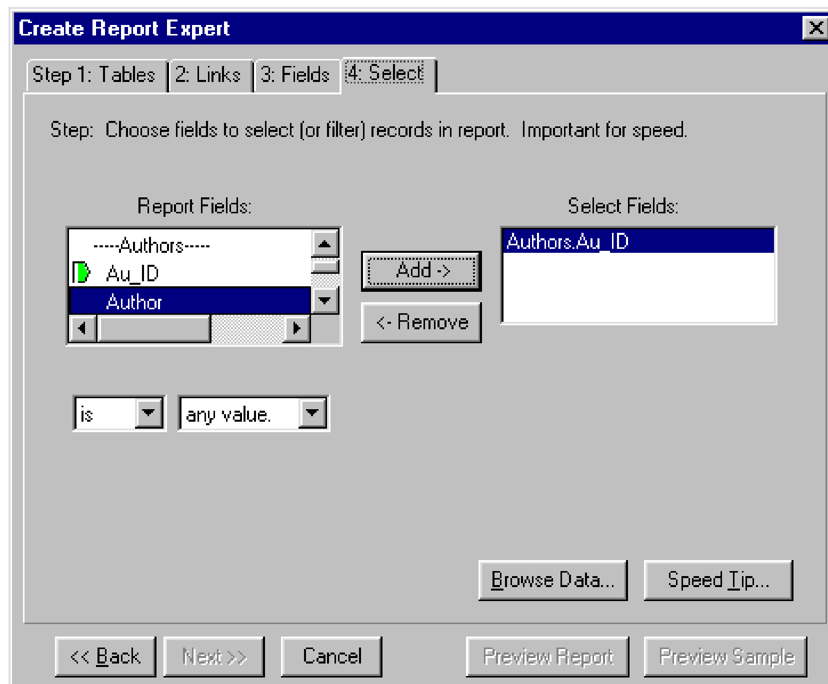
קביעת קריטריון

בכרטיסיה **Select** (תרשים 18.11) ניתן לקבוע קריטריון להצגת הרשומות שיופיעו בדוח. לדוגמה, בדוח המציג רשימת עובדים ניתן לקבוע שהדוח יציג רק את העובדים בטווח גיל מסוים, או רק את העובדים בעלי 3 ילדים ומעלה וכדו'.



תרשים 18.11

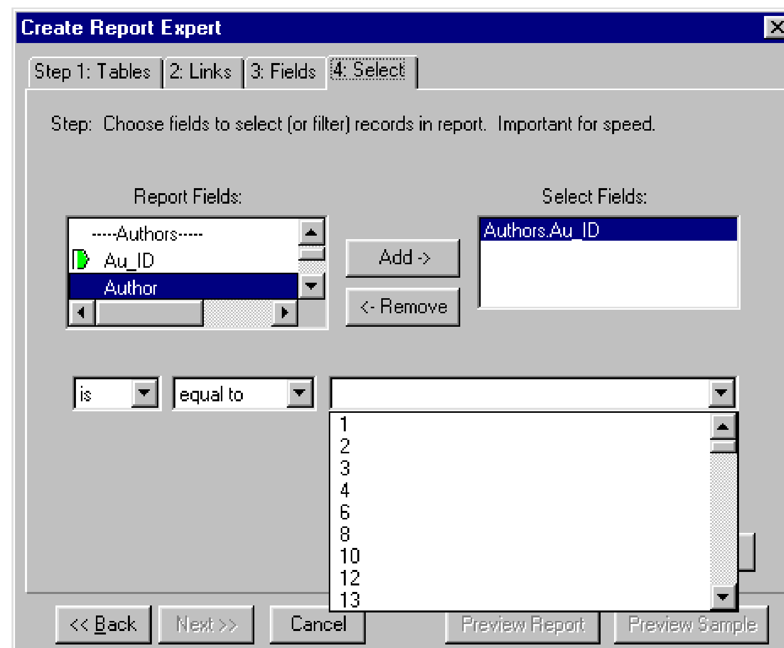
כדי לקבוע קריטריון לדוח, נבחר בשדה מתיבת הרשימה Report Fields, שעליו יופעל הקריטריון ונלחץ על הלחצן **Add** להוספתו לתיבת הרשימה Select Fields. בסיום פעולה זו מוסיף האשף באופן אוטומטי שתי תיבות משולבות (תרשים 18.12) שבעזרתן נקבע את הקריטריון.



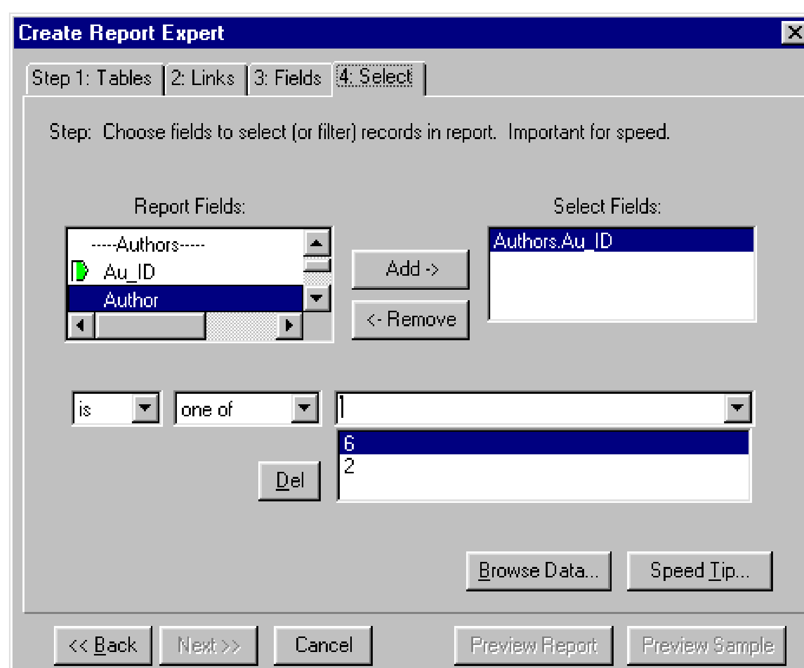
תרשים 18.12

רשימת הקריטריונים כוללת את הקריטריונים הבאים:

- **any value** - הצג את כל הערכים. במילים אחרות, לא נקבע כל קריטריון.
- **equal to** - הצג רק את הערכים השווים לערך מסוים. במצב זה מוצגת אוטומטית תיבה משולבת נוספת המכילה את כל הערכים הקיימים בשדה הקריטריון (תרשים 18.13). מתוך תיבה זו בחר בערך הרצוי.
- **one of** - הצג רק את הערכים השווים לאחד מהערכים שבקבוצת הערכים. גם כאן תופיע תיבה משולבת שמציגה את כל הערכים הקיימים של השדה. תופיע גם תיבת רשימה שאליה מועתקים הערכים שנבחרו מתוך התיבה המשולבת (תרשים 18.14). שים לב שבעזרת הלחצן **Del** ניתן למחוק ערכים מקבוצת הערכים המוצגים בתיבת הרשימה.



תרשים 18.13



תרשים 18.14

- **less than** - מציג רק את הערכים הקטנים מערך מסוים. שים לב שאם בחרת בערך 10 יופיעו הערכים **הקטנים** ממנו, הווה אומר 1-9, בלבד, הערך 10 במקרה זה **לא** ייכלל. כדי לכלול את הערך 10 סמן ✓ בתיבת הסימון **or equal to**.
- **greater than** - מציג רק את הערכים הגדולים מערך מסוים. גם כאן, צריך לבחור באפשרות **or equal to** כדי לכלול את הערך עצמו.
- **between** - מציג את הערכים בטווח מסוים. בחירת "**between 1 and 100**" למשל (תרשים 18.15), תציג את כל הערכים בטווח זה **כולל** הערכים 1 ו- 100.

Create Report Expert
Step 1: Tables | 2: Links | 3: Fields | 4: Select
Step: Choose fields to select (or filter) records in report. Important for speed.
Report Fields: -----Authors----- Au_ID Author
Add -> | <- Remove
Select Fields: Authors.Au_ID
is | between | 1
and
100
Browse Data... | Speed Tip...
<< Back | Next >> | Cancel | Preview Report | Preview Sample

תרשים 18.15

- **formula:** - מציג רק את הערכים שמתאימים לנוסחה מסוימת. לדוגמה, הנוסחה "Length ({Authors.Author})<=10" גורמת להצגת רשומות מחברי ספרים, שאורך המחרוזת של שמם קטן מ-10 תווים או שווה להם. על נוסחאות ועורך הנוסחאות נרחיב את הדיבור בהמשך.

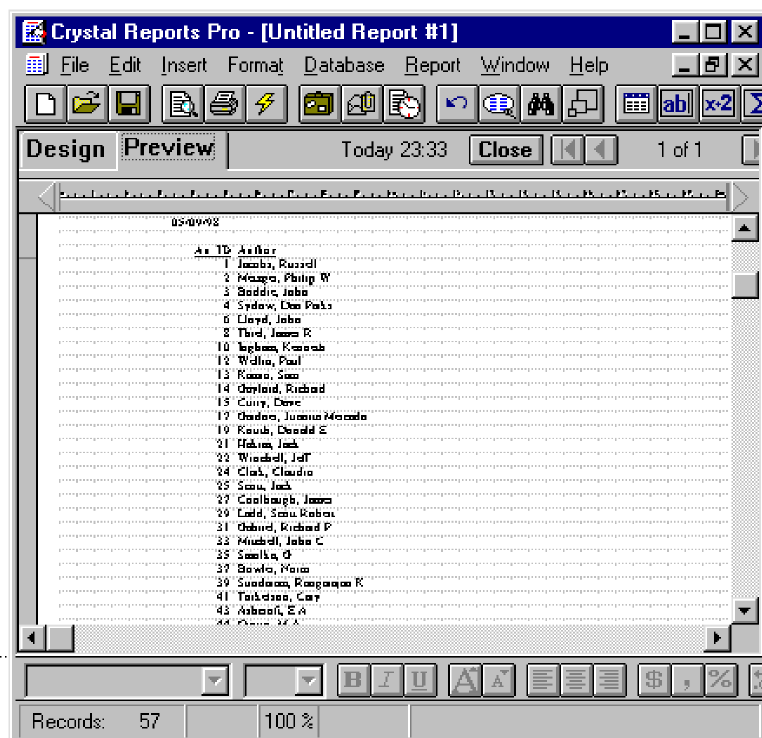
שים לב שניתן לקבוע קריטריונים ליותר משדה אחד. ניתן לקבוע קריטריון גם בהמשך, אם הדבר לא נעשה בשלב זה.

בסיום שלב זה נלחץ על **Preview Report** ("הצג לפני הדפסה") כדי להציג את הדוח כדי לראות את העיצוב, לפני שהוא מועבר להדפסה. אם הדוח המוצג אינו קריא בגלל

גודל התצוגה אפשר לשנות את המיקוד בעזרת לחצן הזום [icon] הכולל שלושה מצבי מיקוד.

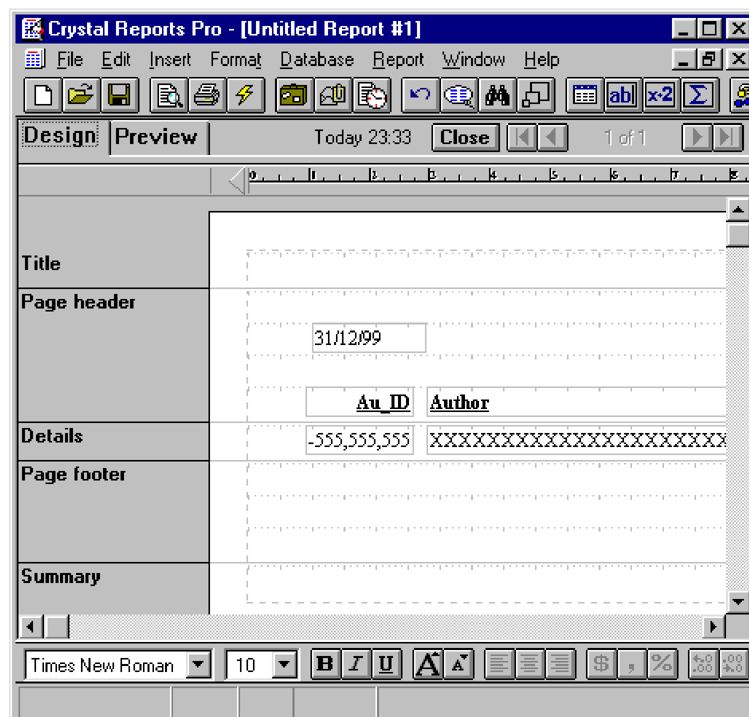
מבט עיצוב (Design)

הדוח המוצג בכרטיסיה **Preview** הוא הדוח שיוצג למשתמש ושיודפס למדפסת (תרשים 18.16).



תרשים 18.16

הכרטיסיה **Design** מציגה את אותו דוח בתצוגת עיצוב (תרשים 18.17), כדי שאפשר יהיה להכניס בו שינויי עיצוב.



תרשים 18.17

חלקי הדוח (Sections)

באופן רגיל הדוח מחולק לחמישה חלקים, הקרויים Sections, שכל אחד מהם מייצג אזור אחר בדוח. חמשת החלקים הם:

1. **Title** - כותרת הדוח. בחלק זה יימצא בדרך כלל לוגו החברה והכותרת המתארת את תוכן הדוח. כל מה שנמצא בחלק זה של הדוח יופיע פעם **אחת** בלבד בראש הדוח, ולעיתים גם בדף נפרד.
2. **Page header** - ראש עמוד. כל מה שיופיע בחלק זה של הדוח יוצג בראש **כל** עמוד בדוח, כמו לדוגמה שם הדוח (במלואו, או מקוצר), תאריך, ולעיתים גם מספר עמוד.
3. **Details** - פרטי הדוח. חלק זה כולל את פרטי כל רשומה הכלולה בדוח. כל מה שנמצא בחלק זה יוצג בכל רשומה בדוח.
4. **Page footer** - תחתית עמוד. כל מה שיופיע בחלק זה של הדוח יוצג בסוף **כל** עמוד בדוח. שדה מספר עמוד יופיע בדרך כלל בחלק זה של הדוח. לעיתים קובעים את מספר העמוד בראש העמוד.

5. **Summary** - סיום הדוח. חלק זה יופיע פעם אחת בדוח, בסופו. לעיתים הוא יוצג בדף נפרד.

עריכת שדה

בתצוגת עריכה של הדוח ניתן למחוק, להוסיף, או לשנות מראה, מיקום וסגנון של השדות המופיעים בדוח.

לחיצה ימנית על השדה מציגה תפריט מקוצר הכולל אפשרויות עריכה של השדה, כשינוי צבע סגנון וכו׳. ניתן לערוך שדה ישירות מסרגל הגופנים הנמצא בתחתית הדוח (תרשים 18.18). אם הסרגל אינו מוצג, לחיצה ימנית על סרגל הכלים ובחירה באפשרות **Show Font Bar** תגרום להצגתו.



תרשים 18.18

כדי למחוק שדה, נבחר בו על ידי לחיצה עליו, ונקיש על Delete.

עורך הנוסחאות

שדה נוסחה מיוצג בתו @ משמאל לשמו. כדי לערוך שדה כזה נפעיל את **עורך הנוסחאות** על ידי לחיצה ימנית על השדה ובחירה באפשרות **Edit Formula**. תרשים 18.19 מציג את עורך הנוסחאות.

כתיבת נוסחה

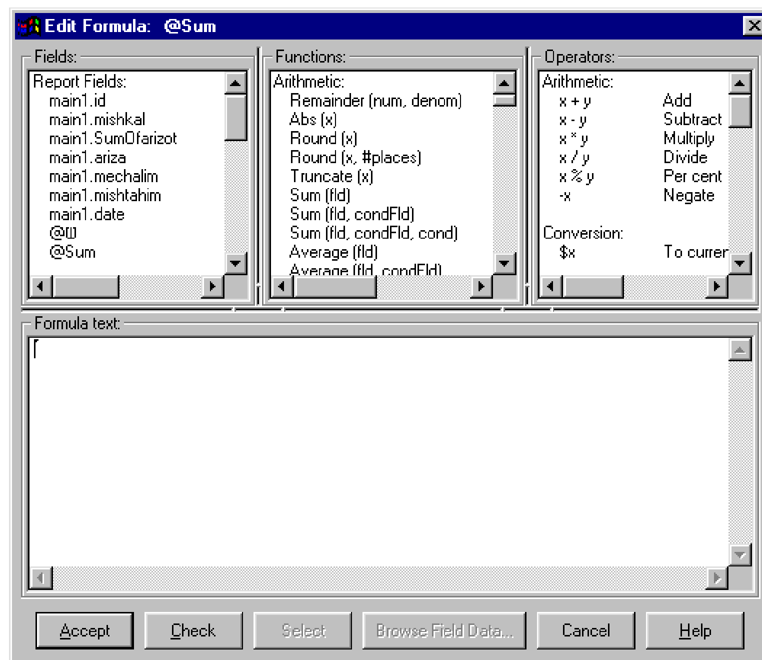
חלון הנוסחאות מחולק לארבעה תת-חלונות. החלון השמאלי **Fields** מציג רשימה הכוללת את כל השדות שבדוח, את הנוסחאות ושאר השדות של מסד הנתונים, אף אלה שאינם כלולים בדוח. מחלון זה נבחר את השדה אשר ייכלל בנוסחה.

החלון **Functions** שלימינו (האמצעי) כולל את רשימת הפונקציות שמחולל הדוחות מכיר ואשר ניתן להשתמש בהן בבניית הנוסחה.

החלון הימני ביותר **Operators** כולל את כל האופרטורים שניתן להשתמש בהם בעורך הנוסחאות.

אין חובה להשתמש בחלונות אלה לבניית הנוסחה ומטרתם להקל על העבודה בלבד. מי שיודע לכתוב את הנוסחה לבד, יוכל לפנות לחלון **Formula text** שבתחתית החלון המוצג, כדי לכתוב בו את הנוסחה.

עורך הנוסחאות מאפשר לבדוק את נכונות הנוסחה ותקינותה. לשם כך נלחץ על הלחצן **Check**. אם יש שגיאה בנוסחה העורך יתריע על כך (תרשים 18.20). ההודעה שתתקבל תציג את מהות השגיאה, והסמן יעמוד במקום בו אותרה השגיאה בנוסחה.

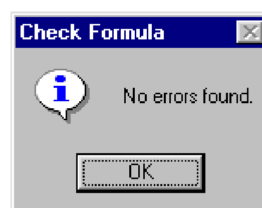


תרשים 18.19



תרשים 18.20

כאשר העורך מאשר את תקינות הנוסחה (תרשים 18.21) צריך ללחוץ על **Accept** לאישור, ואחר כך צריך למקם אותה במקום הרצוי בדוח.



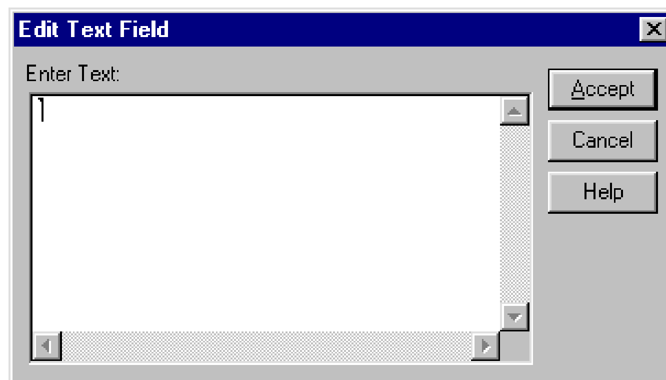
תרשים 18.21

הוספת שדה

ניתן להוסיף לדוח שדות לאחר יצירת הדוח לראשונה. יש שדות מיוחדים שניתן להוסיף כשדה תאריך, שדה המציג את מספר העמוד וכדו׳. כמו כן ניתן להוסיף שדות ממסד הנתונים ושדות עזר אחרים.

שדה טקסט

תוכל להוסיף שדה טקסט לדוח. שדה זה שימושי בדרך כלל לנתינת כותרת לדוח, או למתן הוראות וכו׳. כדי להוסיף שדה טקסט לדוח בחר באפשרות **Text Field** שבתפריט **Insert** או לחץ על שבסרגל הכלים. בחירה זו תגרום להופעת החלון **Edit Text Field** בו תכתוב את הטקסט הרצוי (תרשים 18.22). כל שנותר הוא למקם את השדה במקום הרצוי בדוח.

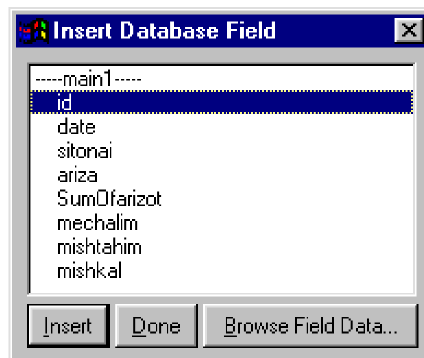


תרשים 18.22

שדה ממסד הנתונים

למרות הקביעה הראשונה איזה שדות ייכללו בדוח, ניתן בכל זמן למחוק ולהוסיף שדות ממסד הנתונים. להוספת שדה ממסד הנתונים נבחר באפשרות **Database Field** שבתפריט **Insert** או נלחץ על שבסרגל הכלים.

בחירה זו תגרום להופעת החלון **Insert Database Field** (תרשים 18.23). נבחר בשדה הרצוי ונלחץ על **Insert** כדי למקם אותו בדוח. בסיום נלחץ על **Done**. באפשרותך להוסיף יותר משדה אחד, לחיצה על Done תיעשה לאחר שילוב של כל השדות.



תרשים 18.23

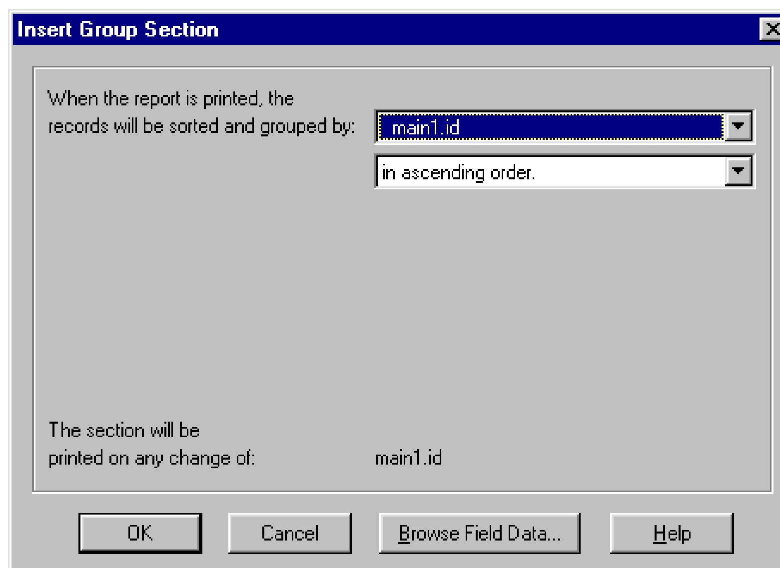
פעולה זו היא הוספת שדות קיימים מתוך מסד הנתונים המקושר לדוח. **להוספת** מסד נתונים חדש לדוח צריך לבחור באפשרות **Add Database to Report** מתפריט **Database**.

שדה מקבץ (Group By)

ניקח לדוגמה דוח המציג את רשימת כל העובדים במפעל. דוח זה יציג ברצף אחד את הרשימה עם קריטריון, או ללא התייחסות לקריטריון מסוים. מה נעשה כאשר נתבקש להציג את אותה רשימת עובדים, אך הפעם לחלק אותה לפי מחלקות. כלומר, אלה שבמחלקת כספים יוצגו בקבוצה אחת, עובדי הנהלה בקבוצה שנייה וכו'.

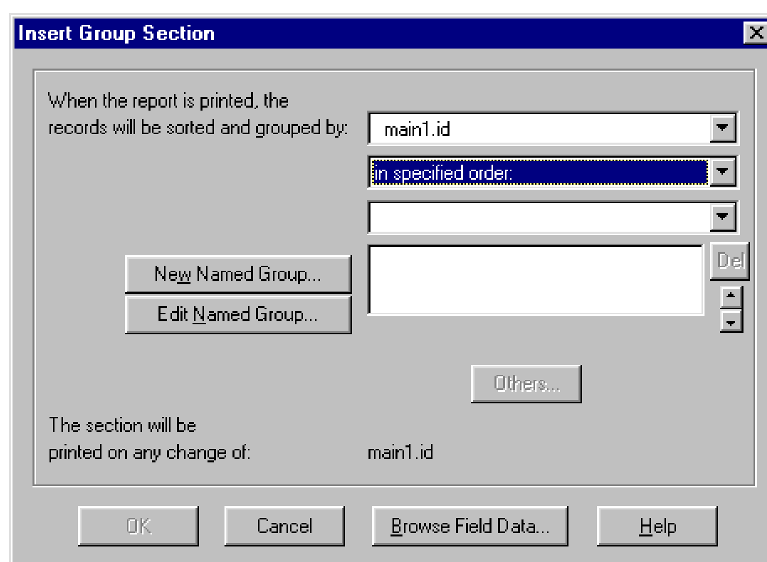
במקרה זה ניעזר בשדה מקבץ. השדה המקבץ בדוגמה זו יהיה, באופן טבעי, המחלקה אליה שייך כל עובד. כיצד נציין את הדרישה להקבצת הנתונים לפי שדה מסוים?

נבחר באפשרות **Group Section** בתפריט **Insert**. בחירה זו תציג את החלון **Insert Group Section** (תרשים 18.24).



תרשים 18.24

בחלון זה יש שתי תיבות משולבות; האחת (העליונה) לבחירת השדה המקבץ, והשנייה שמתחתיה קובעת את אופן ההקבצה. כדי לחלק את הקבוצות לתת-קבוצות נבחר באפשרות השלישית **in specified order** ונבנה את תת-הקבוצות (תרשים 18.25).



תרשים 18.25

בצד שמאל של הדוח שמוצג מבט עיצוב, בחלק Sections, נוסף החלק המקבץ.

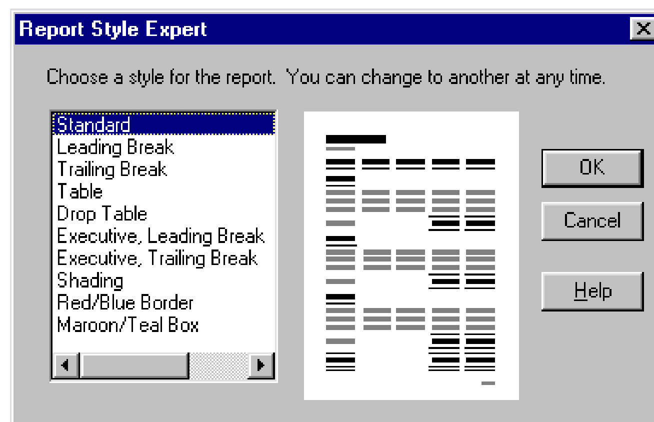
אין הגבלה במספר השדות המקבצים שניתן להוסיף לדוח, אך כל חלק מקבץ נוסף יהווה תת-הקבצה בתוך החלק המקבץ שלפניו.

קביעת סגנון דוח

לאחר סיום בניית הדוח, מחולל הדוחות מאפשר בחירת סגנון לדוח מתוך רשימת סגנונות. אין חובה להשתמש באחד הסגנונות האלה, אך אם אחד מהם נמצא מתאים, אפשר להחיל אותו על הדוח.

החלת סגנון קיים על הדוח נעשית באופן פשוט. נבחר באפשרות **Report Style Expert** מתפריט **Format**. החלון Report Style Expert נפתח (תרשים 18.26), מתוכו נבחר בסגנון הרצוי ונאשר בלחיצה על OK.

לסיום יש לשמור את הדוח ולקבוע לו שם. הדוח נשמר כקובץ עצמאי עם סיומת **rpt**.



תרשים 18.26

הצגת הדוח מתוך ויז'ואל בייסיק

הדוח שנוצר בעזרת מחולל הדוחות הוא דוח עצמאי ואינו תלוי ביישום ויז'ואל בייסיק. כדי לאפשר למשתמש להפעיל את הדוח מתוך היישום, צריך ליצור ביניהם קישור.

קישור הדוח אל היישום נעשה בעזרת הפקד **Crystal Report 4.6**. פקד זה אינו חלק מהפקדים הסטנדרטיים וצריך להוסיף אותו לארגז הכלים באופן ידני. לאחר הוספת הפקד לארגז הכלים מציבים אותו בטופס כפי שנוהגים עם כל פקד אחר. מיקום הפקד בטופס אינו עקרוני, מכיון שהוא לא נראה למשתמש. לאחר הוספת הפקד לטופס, הוא יוצר את הקשר אל הדוח.

המאפיין ReportFileName

כדי לקשר את היישום עם הדוח בעזרת הפקד, צריך להודיע לפקד לאיזה דוח עליו ליצור את הקישור מתוך היישום. המאפיין של הפקד לקביעת הקשר מהיישום אל קובץ הדוח הוא **ReportFileName**. הערך שיקבל מאפיין זה הוא שם קובץ הדוח ונתיבו. לדוגמה:

```
rptMyReport.ReportFileName = "C:\Reports\MyReport.rpt"
```

אין צורך להוסיף פקד לכל דוח; פקד אחד יכול להציג את כל הדוחות ביישום. עם זאת, בכל פעם צריך לשנות את שם הקובץ של הדוח שהפקד מציג, כדי שיציג את הדוח המתאים.

המאפיין DataFileName

קשר חשוב נוסף שיש לבצע עם הפקד הוא הקשר למסד הנתונים, או למסדי נתונים, שעליהם מבוסס הדוח. המאפיין המטפל בקשר זה הוא **DataFileName**. לדוגמה:

```
rptMyReport.DataFileName(0) = "C:\MyApp\MyDatabase.mdb"
```

מכיון שדוח יכול להיות מקושר למספר מסדי נתונים, המאפיין DataFileName הוא מערך. הקשר למסד הנתונים הראשון יהיה האיבר הראשון במערך DataFileName(0) הקשר למסד הנתונים השני יוגדר DataFileName(1) וכן הלאה.

קביעת יעד לדוח

לאחר קישור הדוח לקובץ ולמסד הנתונים המתאים, יש לקבוע יעד למשלוח דוח. שני היעדים העיקריים אליהם אפשר לשלוח את הדוח הם למסך ולמדפסת.

המאפיין האחראי על קביעת היעד של הדוח הוא **Destination**. לדוגמה:

```
rptMyReport.Destination = crptToWindow
rptMyReport.Destination = crptToPrinter
```

הדפסת הדוח

לבסוף, כל שנותר הוא **להדפיס** את הדוח. "הדפסת דוח" היא ביטוי כללי להפקה, שיכולה להיות למדפסת או למסך למשל.

השירות האחראי על הדפסת הדוח (ליעד שנקבע לו) הוא **PrintReport**. לדוגמה:

```
rptMyReport.PrintReport
```

מה קורה כשאין מדפסת

אם הדוח נשלח למדפסת ואין מדפסת, או היא אינה מחוברת או שאינה מוגדרת, אל דאגה. היישום לא יעצור עם הודעת שגיאה. לשגיאה מסוג זה אחראית מערכת ההפעלה והיא תיתן הודעה מתאימה למשתמש עם מהות השגיאה.

בחירת הרשומות שיוצגו בדוח

למדנו כיצד נוכל לקבוע בהגדרת הדוח קריטריון לבחירת הרשומות שתוצגנה בו. לדוגמה, נרצה להציג את רשימת העובדים בגיל 35 ומעלה. אולם אפשר לבנות דוח כללי שיציג את כל הרשומות, ומתוך היישום בוויז'ואל בייסיק נקבע את החתך לרשומות שיוצגו כפלט של הדוח.

בחירת חתך הרשומות לתצוגה מתוך היישום עדיפה לעיתים, מכיון שניתן להשתמש בדוח אחד גלמי וממנו לגזור נתונים, או דוחות, בחתכים שונים.

המאפיין שקובע את החתך לדוח הוא **SelectionFormula**. לדוגמה:

```
rptMyReport.SelectionFormula = "{Employees.Age} >= 35"
```

בדוגמה זו הקריטריון מתייחס לשדה Age מטבלת Employees שמוגדר כך: {Employees.Age}. כל רשומה שבה ערך השדה גדול או שווה ל- 35 תופיע בדוח המופק. שאר הפרטים בדוח אינם משתנים, אלא שמשפט זה קובע את התנאי להצגת הרשומות מתוך הדוח הגולמי שהוכן.

תרגול

1. בנה דוח המציג את רשימת כל הסטודנטים באוניברסיטה.
2. בנה דוח המציג את רשימת הסטודנטים לפי פקולטות. הדוח יציג סיכום ביניים של מספר התלמידים בכל פקולטה. בסוף הדוח יוצג סיכום כללי של מספר כל הסטודנטים באוניברסיטה.

19 : פונקציות dll

פעמים רבות אנו נתקלים בפני אתגרי תכנות המחייבים פתרון הכולל שורות קוד רבות ולא מעט חשיבה מקורית. בחלק מהמקרים אף אין בידינו כלים לפתור את הבעיה הניצבת לפנינו.

כאן נכנסות לתמונה פונקציות **API** או פונקציות **DLL**, המספקות מיגוון כלים לעזרת המתכנת, כמו לדוגמה, מציאת ספריית המערכת במחשב בו פועל היישום. לשם הבהרה, **קובץ DLL** הוא קובץ הרצה שיכול להכיל בתוכו **פונקציית API** אחת או יותר וגם קוד נוסף.

לרוב נהוג לקרוא לספריית המערכת בשם "Windows", או "Win95" ויש הקוראים לה "Windows95". כיצד נזהה אותה אם יש למשל שתי ספריות, אחת "Windows" והשנייה "Win95", איזו מהן היא ספריית המערכת שלנו? במקרה כזה חבל להשקיע מאמץ בבדיקת הספריות וחקירתן, כי עלינו רק לקרוא לפונקציית API המתאימה שתבצע עבורנו את העבודה.

השימוש בפונקציית API (או DLL) פשוט מאוד. מצהירים פעם אחת על הפונקציה, ובכל פעם שנרצה להשתמש בשירותיה, נקרא לה (כפונקציה רגילה) עם הארגומנטים המתאימים. בהמשך לדוגמה הקודמת, כדי לדעת מהי ספריית המערכת נקרא לפונקציית DLL בשם GetWindowsDirectory.

מכלול פונקציות API מקיף כמעט כל נושא שמתכנתים נזדקקים לו. לנוחותנו מספקת לנו ויז'ואל בייסיק את **API Text Viewer** המכיל מידע אודות פונקציות אלו. יישום זה הוא מכלול כלים: תוכנית שירות לחיפוש ברשימת הפונקציות, העתקת תחביר הפונקציה אל ויז'ואל בייסיק, והעיקר – פירוט הפרמטרים שדרושים לכל פונקציה.

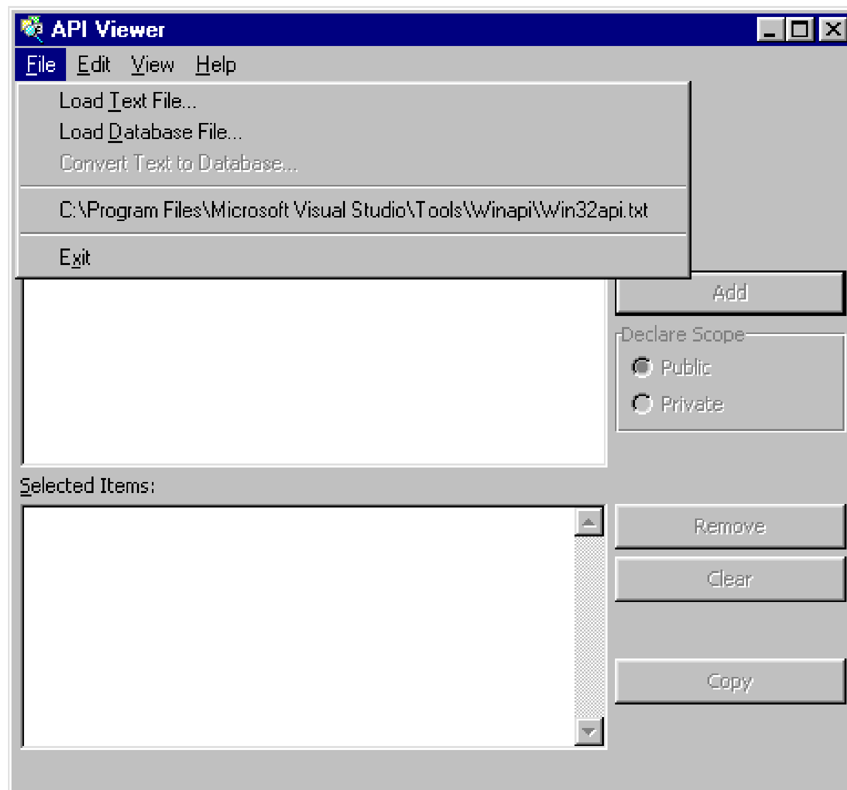
API Text Viewer

היישום **API Text Viewer** הוא חלק מקבוצת היישומים בה מותקנת ויז'ואל בייסיק. לחץ על לחצן **התחל**, ומתפריט **תוכניות** בקבוצת היישומים **Microsoft Visual Studio 6** בחר באפשרות **Microsoft Visual Studio 6 Tools** ושם בחר באפשרות **API Text Viewer**.

API Text Viewer לוקח קובץ טקסט המכיל את רשימת כל הפונקציות והפרמטרים שמקבלת כל פונקציה ומציג את רשימת שמות הפונקציות. כשנבקש לחפש פונקציה מסוימת, הוא יעשה זאת עבורנו בתוך אותו קובץ טקסט. לכן, בשלב ראשון יש לטעון אל API Text Viewer את קובץ הטקסט המכיל את רשימת הפונקציות.

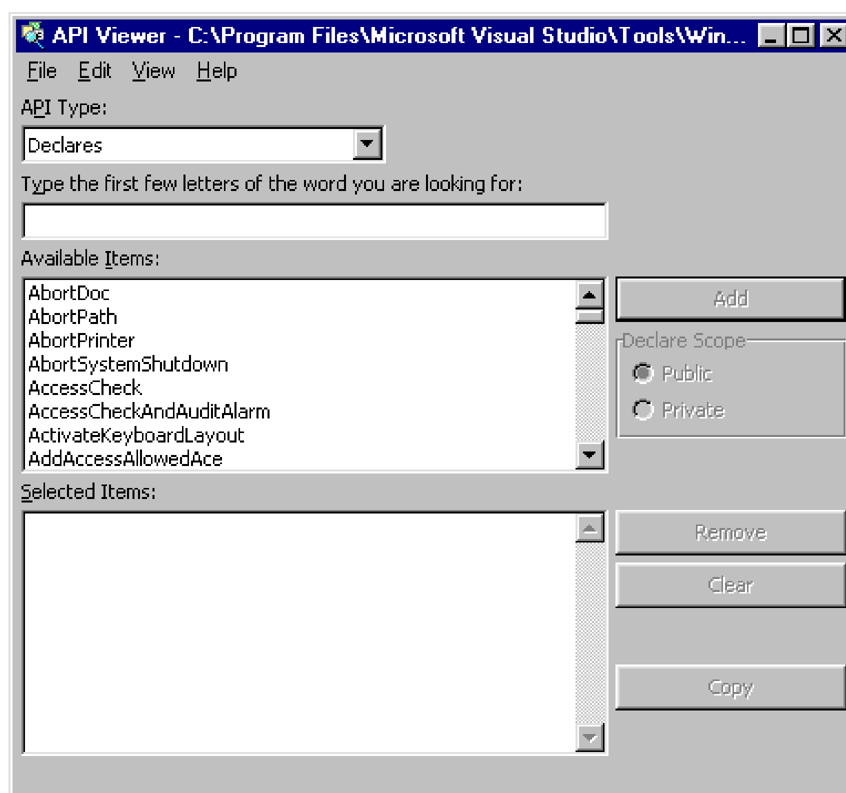
הקובץ Win32api.txt

הקובץ Win32api.txt מכיל את רשימת פונקציות DLL. נטען אותו אל API Text Viewer. כדי לבצע את הטעינה בחר באפשרות **Text File Load** שבתפריט **File** ובחר בקובץ Win32api.txt (תרשים 19.1).



תרשים 19.1

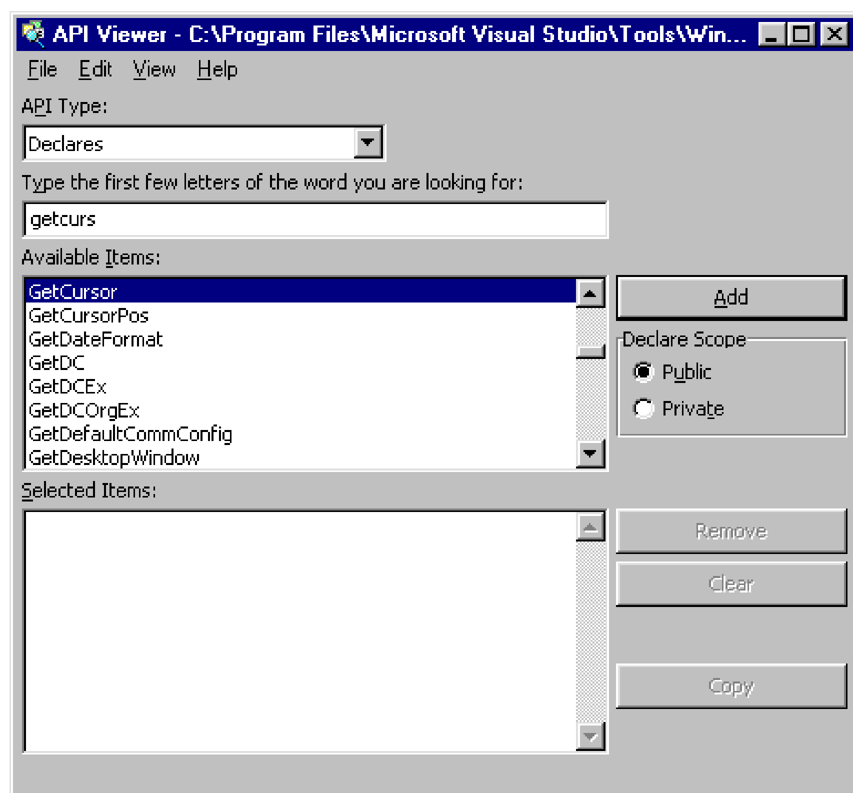
הקובץ נטען ל-API Text Viewer ובסיום הטעינה מוצגת רשימת שמות הפונקציה בתיבת הרשימה **Available Items** (תרשים 19.2).



תרשים 19.2

חיפוש פונקציה

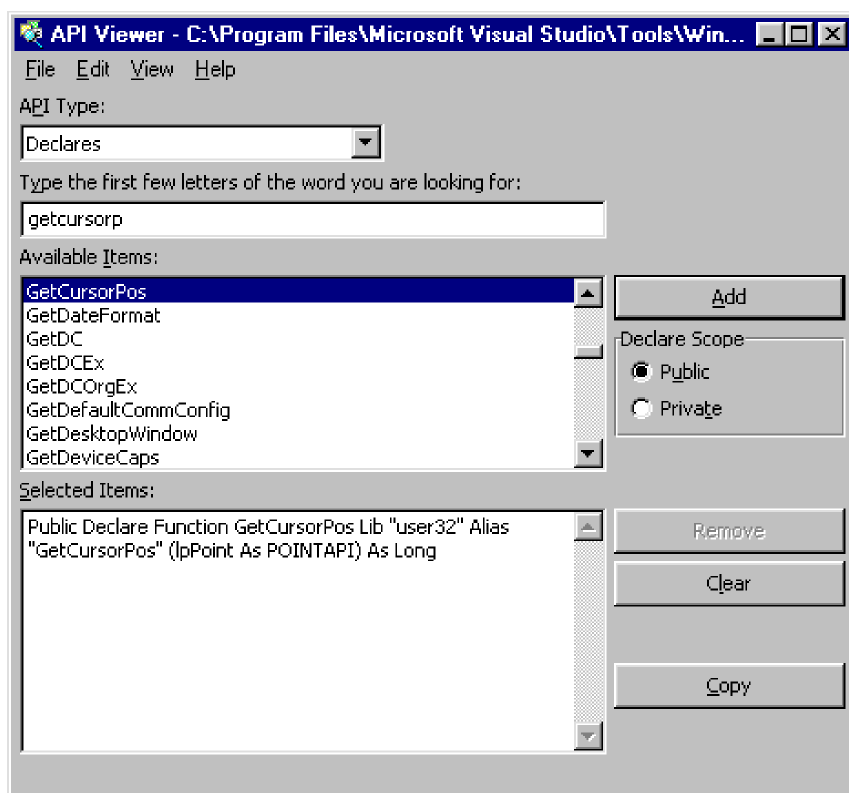
אם אנו מכירים את הפונקציה שבה אנו רוצים להשתמש וזוכרים את שמה, ניתן לגלול את הרשימה ולבחור ישירות בפונקציה. לפעמים אנו זוכרים את תחילת שם הפונקציה בלבד. API Text Viewer מספק לנו שירות חיפוש לפי מחרוזת חלקית של שם הפונקציה. לביצוע החיפוש הקלד בתיבת הטקסט הראשונה "Type the first few letters..." את שם הפונקציה או חלק ממנה וה- API Text Viewer יחפש בהתאם להקלדה שלך את הפונקציה המתאימה (תרשים 19.3).



תרשים 19.3

הצגת הפונקציה

הרשימה Available Items כוללת את שם הפונקציה בלבד. כדי לצפות בהצהרת הפונקציה במלואה לחץ על לחצן **Add**. הלחיצה גורמת לכתיבת ההצהרה על הפונקציה בתיבת הרשימה **Selected Items**. הצהרה זו היא ההצהרה בה אנו משתמשים בשורות הקוד בתוכנית והיא כוללת את רשימת הפרמטרים של הפונקציה. מלבד ההצהרה על הפונקציה, שורה זו מזכירה לנו איזה פרמטרים מקבלת הפונקציה ואת סוג המשתנה של כל אחד מהם (תרשים 19.4).



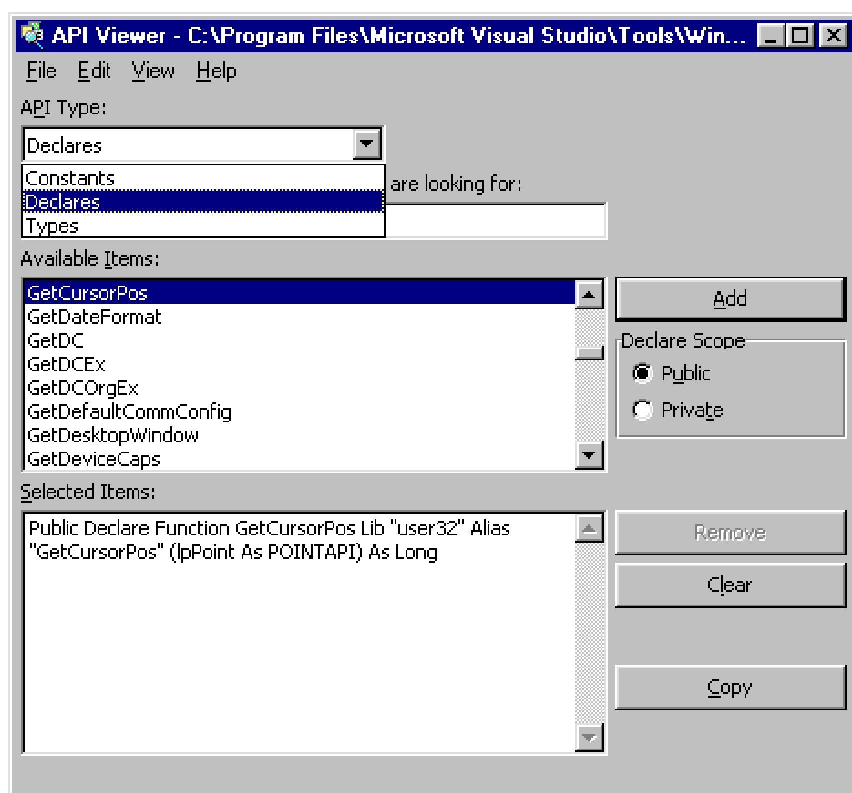
תרשים 19.4

מבנים (Types)

אם רשימת הפרמטרים שמקבלת הפונקציה כוללת את סוגי המשתנים הסטנדרטיים כ- Integer, String וכו', שורת ההצהרה המופיעה בתיבת הטקסט Selected Items מספיקה. אולם לפעמים כוללת רשימת הפרמטרים משתנה שהוא מסוג **מבנה** (Data Type) בוויז'ואל בייסיק. במקרה כזה, לא מספיק להעתיק לתוכנית את שורת ההצהרה על הפונקציה, אלא דרוש להצהיר גם על המבנה בו נעשה שימוש בפונקציה.

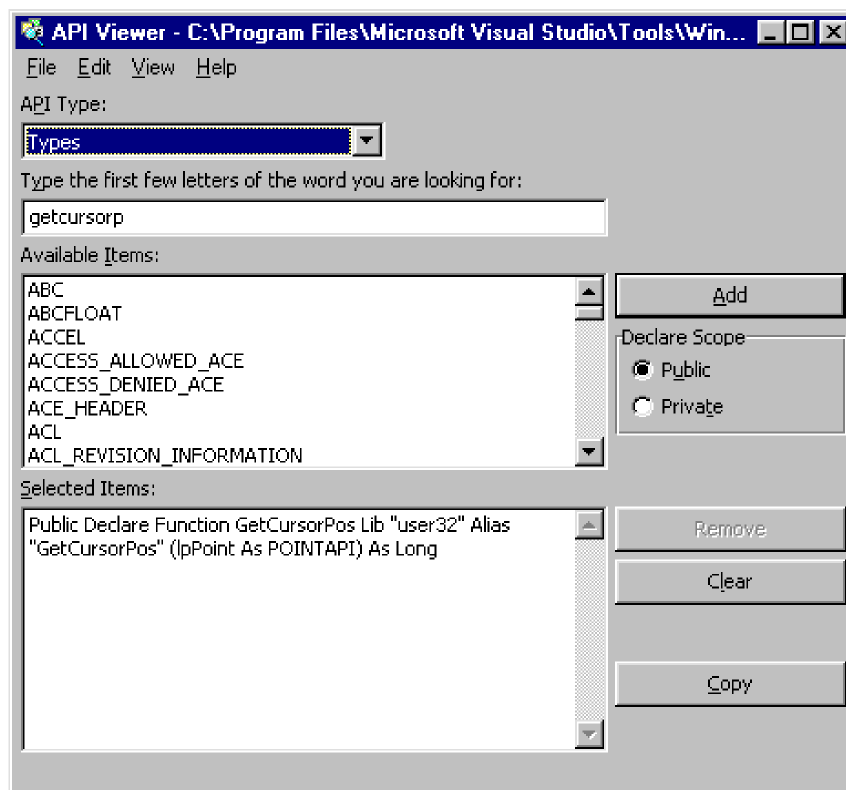
API Text viewer מספק רשימה נוספת (Types) המכילה את כל סוגי המבנים, בדומה לרשימת הפונקציות עצמן. גם ברשימה זו ניתן לבצע חיפוש בעזרת Search, למצוא את המבנה הדרוש ולהציגו בצורה מלאה על כל משתניו בתיבת הטקסט Selected Items.

כדי להציג את רשימת המבנים בתיבת הרשימה Available Items, בחר מתוך התיבה הנפתחת API Type באפשרות **Types** (תרשים 19.5).



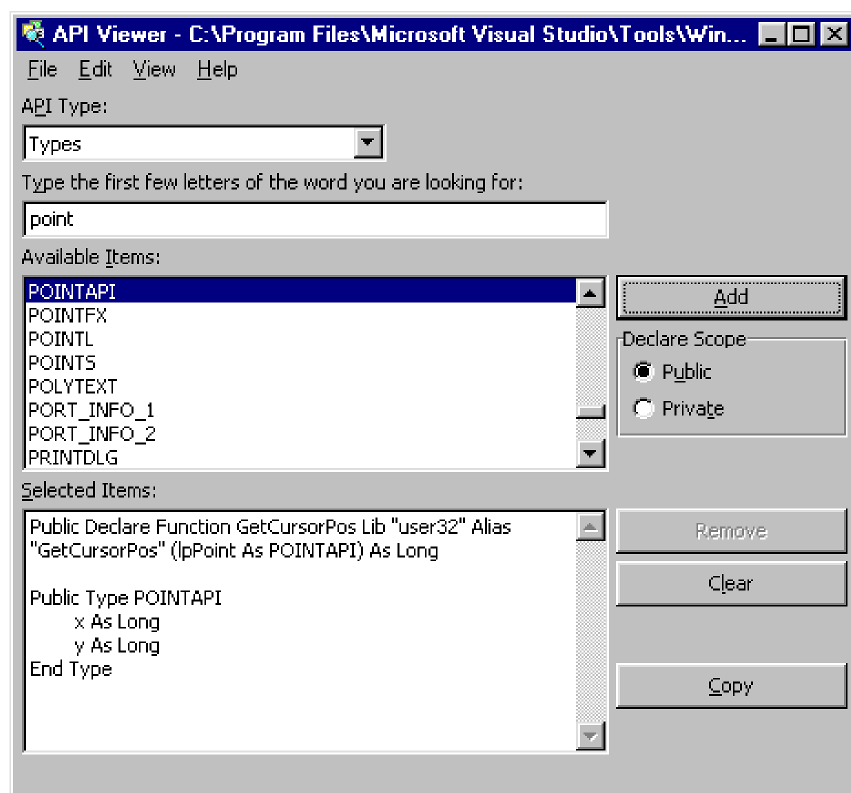
תרשים 19.5

אפשרות זו מחליפה את רשימת הפונקציות ברשימת המבנים (תרשים 19.6).



תרשים 19.6

לדוגמה, הפונקציה **GetCursorPos** מקבלת כפרמטר משתנה מסוג המבנה POINTAPI. בחר מהתיבה הנפתחת **API Type** באפשרות **Types** וחפש את המבנה POINTAPI. הוסף את המבנה בעזרת הלחצן **Add** אל תיבת הטקסט **Selected Items** בסמוך להצהרה על הפונקציה (תרשים 19.7). מלבד ההצהרה על הפונקציה תתבצע הצהרה על המבנה בו ייעשה שימוש במסגרת הפונקציה.



תרשים 19.7

העתקת הפונקציה לוויז'ואל בייסיק

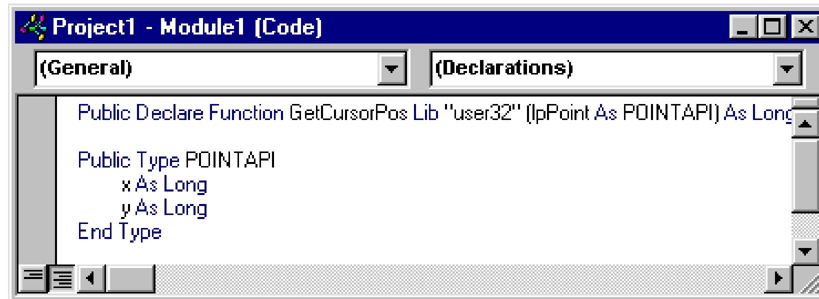
לאחר בחירת הפונקציה הרצויה, ובמקרה הצורך במבנה או במבנים רצויים, והצגתה בתיבת הטקסט **Selected Items**, כל שנותר הוא להעתיק את שורות ההצהרה לקוד שלנו בוויז'ואל בייסיק.

העתקת ההצהרה על הפונקציה נעשית באופן המקובל ב-Windows: העתקת שורת ההצהרה של הפונקציה אל לוח Windows (Clipboard) ולאחר מכן הדבקת תוכן הלוח בשורת הקוד בוויז'ואל בייסיק (הפעולה Paste).

כדי להעתיק את תוכן תיבת הטקסט Selected Items ללוח, לחץ על הלחצן **Copy**. לאחר מכן עבור לוויז'ואל בייסיק והדבק את תוכן הלוח על ידי בחירה באפשרות **Paste** שבתפריט **Edit**.

הצהרה על הפונקציה במודול

הדבקת (פעולת Paste) הפונקציה בשורות הקוד אינה מוגבלת למקום מסוים. ככלל, ניתן להדביק את ההצהרה גם בתוך שורה הנמצאת בטופס. אולם הצהרה על פונקציית DLL במקום זה תוגדר כ-Private. מקום נוסף בו ניתן להצהיר על פונקציות DLL הוא במודול (תרשים 19.8).



תרשים 19.8

לאחר הצהרה על הפונקציה במודול, טווח ההכרה ואורך החיים שלה הם לכל הפרויקט כולו. הצהרה אחת כוחה יפה לכל התוכנית. טווח ההכרה משתנה בהתאם לקביעה Public או Private שקבענו ב- Declare Scope שב- API Text Viewer. בתרשים זה מוצהרת הפונקציה כ- Public לכן טווח ההכרה שלה יהיה בכל הפרויקט.

קריאה לפונקציה משורות קוד

הצהרה על פונקציה נעשית במודול ופעם אחת בלבד לכל הפרויקט. אולם קריאה לפונקציה נעשית בכל פעם מחדש. בכל פעם שנזדקק לשירותי הפונקציה, נקרא לה.

שים לב שקריאה לפונקציה מסוג DLL זהה לקריאה לפונקציה רגילה שנבנית על ידנו. הפונקציה צריכה לעמוד בכל הכללים החלים על פונקציה רגילה, כמו למשל התאמה של רשימת הפרמטרים לרשימת הארגומנטים הן מבחינת הסדר והן בסוגי המשתנים, ועוד.

לדוגמה, הנה קריאה לפונקציה **GetCursorPos**:

```
Dim dtPnt As POINTAPI
Res = GetCursorPos(dtPnt)
```

הפונקציה מקבלת כפרמטר את כתובת המבנה dtPnt מסוג (POINTAPI) ומחזירה לו את ערכי הקואורדינטות של הנקודה במסך שבהן נמצא העכבר.

פונקציה לדוגמה

לשם הסבר נוסף לשימוש בפונקציות API בשורות קוד בוויז׳ואל בייסיק, נבצע יחד תרגיל המשלב בתוכו שימוש בפונקציית API.

הפונקציה SetWindowText

הפונקציה SetWindowText קובעת את כותרת החלון. היא מקבלת שני פרמטרים: את המצביע/ידית אחיזה (Handle) לחלון לו אנו מעוניינים לקבוע את הכותרת, ומחרוזת (String) המייצגת את הכותרת עצמה. לדוגמה:

```
SetWindowText (Me.hWnd, "My Form")
```

בדוגמה זו שינינו את כותרת החלון הנוכחי (הטופס הנוכחי) ל-MyForm. פעולה זו נעשית על חלון (טופס) ביישום שלנו ללא צורך בפונקציית API, ניתן פשוט לכתוב:

```
Me.Caption = "MyForm"
```

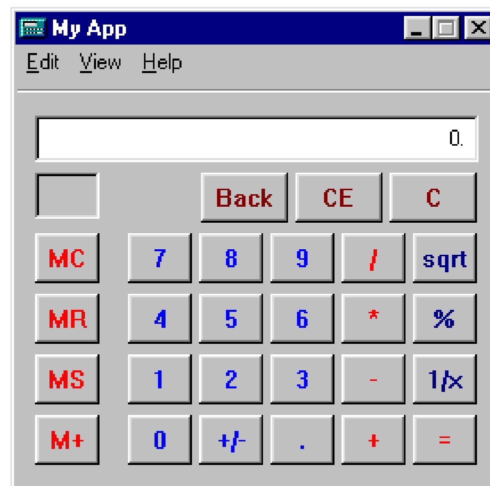
אך מה קורה אם נרצה להתייחס לחלון שאינו חלק מהיישום? לדוגמה, לחיצה על הלחצן בטופס גורמת לפתיחת המחשבון (בתפריט **תוכניות**, **עזרים** שבתפריט **התחל**). במצב רגיל נפתח המחשבון עם הכותרת הרגילה שלו **מחשבון**, אך בעזרת הפונקציה SetWindowText נוכל לשנות את הכותרת שלו לכותרת היישום שלנו, כדי להקנות תחושה למשתמש כאילו המחשבון חלק אינטגרלי מהיישום.

כדי לדעת מהו ערך handle של חלון ה**מחשבון** שנפתח, ניעזר בפונקציה GetWindow. פונקציה זו מחזירה כערך את ה-handle של החלון, אשר יישלח כארגומנט לפונקציה SetWindowText כדי לקבוע לו כותרת. באופן זה נשתמש כדי לקבוע את כותרת חלון המחשבון לכותרת היישום שלנו.

שורות הקוד הבאות נעזרות בשתי פונקציות API כדי להציג כותרת למחשבון:

```
Dim Res, Hnd
Res = Shell("c:\windows\Calc.exe", vbNormalFocus))
Hnd = GetWindow(Me.Hwnd, 3)
Res = SetWindowText(Hnd, "My App")
```

בתחילה התוכנית מפעילה את המחשבון (Calc.exe) בעזרת הפונקציה Shell. פונקציית API – GetWindow – מחזירה כערך את ה-handle של חלון המחשבון שנפתח. ערך זה נשלח כארגומנט לפונקציה SetWindowText הקובעת את הכותרת ״My App״ למחשבון (תרשים 19.9).



תרשים 19.9

תרגול

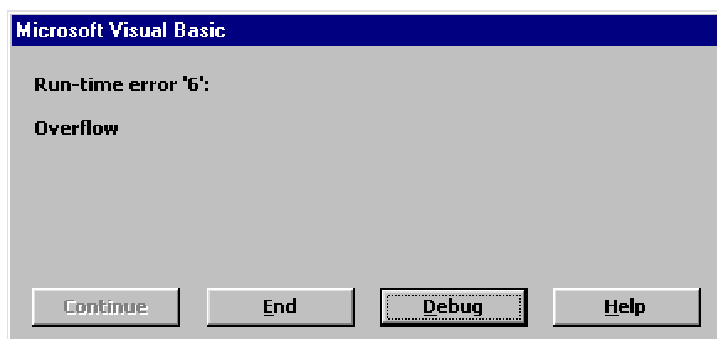
1. צור טופס עם לחצן. לחיצה על הלחצן תגרום לפתיחת הצייר. שנה את כותרת הצייר ל- "הצייר שלי".
2. צור טופס. כל לחיצת עכבר על הטופס תצייר ריבוע בגודל ובמיקום אקראי על הטופס.

הנחיה: היעזר בפונקציית API - Rectangle - ליצירת ריבוע על גבי הטופס.

20: טיפול בשגיאות

במצבים מסוימים מופסקת פעולת התוכנית כתוצאה מטעות שקרתה בה. סוג אחד של טעות, הנובעת באשמת מתכנתים, היא הטעות הקרויה **באג**. טעות זו, כשהיא קורית במהלך פעולת התוכנית, גורמת לקריסת התוכנית ולהפסקת פעולתה. הפתרון היחיד במצב זה הוא לבדוק את שורות הקוד של התוכנית כדי לפענח את סיבת מופע הטעות, הבאג, ותיקון. לפעמים הבאג נמצא מהר והתיקון פשוט ומהיר, ולעיתים הטעות היא לוגית והמעקב אחריה נמשך זמן רב יותר. בכלים העומדים לרשותנו למעקב אחר באגים מסוג זה נדון מאוחר יותר בפרק זה, בסעיף הדן ביישום Debugger.

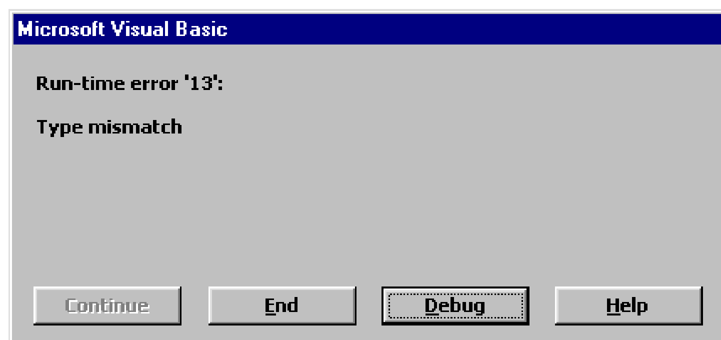
שגיאה מסוג אחר נובעת משימוש לא חוקי של המשתמש ביישום. לדוגמה, המשתמש הקליד ערך מספרי גדול למשתנה שיכול להכיל ערך מספרי קטן (תרשים 20.1).



תרשים 20.1

או למשל, המשתמש הקליד ערך מסוג מסוים (נניח String) למשתנה מסוג אחר (למשל Integer) ופעולה זו גרמה לשגיאה (תרשים 20.2).

בדוגמה נוספת אנו מאפשרים למשתמש לבחור קובץ ולפתוח אותו. כאשר המשתמש בוחר קובץ שנמצא בכונן :A ומורה לפתוח אותו, התוכנית מחפשת את הקובץ בכונן :A. אולם, כאשר אין דיסקט בכונן :A נגרמת שגיאה המפסיקה את פעולת התוכנית. כדי למנוע שגיאות מסוג זה, נדאג לטפל (בעזרת קוד) במקומות בהם הסיכוי לטעות גבוה. כמובן שעל המתכנתים מוטל לצפות צעד אחד קדימה ולהקדים תרופה למקומות בהם המשתמש עלול לטעות, ועליהם למנוע מראש את השגיאה הצפויה על ידי מתן פתרון מתאים בקוד.



תרשים 20.2

המשפט On Error

כדי ללכוד שגיאה ולטפל בה באופן מוגדר, בלי לתת לה את האפשרות להפסיק את פעולת התוכנית, נשתמש במשפט **On Error**. משפט זה מפנה את התוכנית לקטע קוד המטפל בשגיאה. לדוגמה,

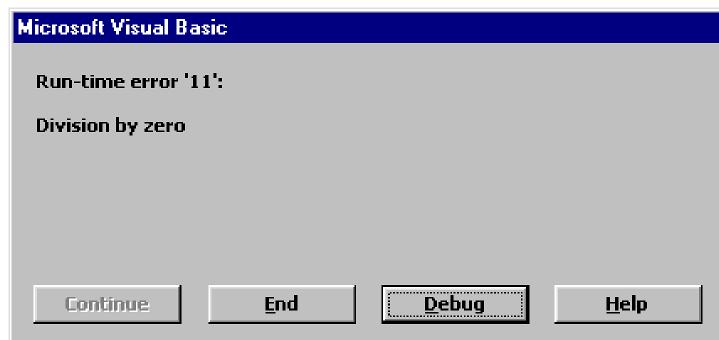
```
Private Sub cmdCalc_Click()
   On Error GoTo ErrHandler
   Dim intDev As Integer
   intDev = Val(Text1.Text) / Val(Text2.Text)
   Exit Sub

ErrHandler:
   MsgBox "You can't division by zero!"
   text2.SetFocus
End Sub
```

שים לב שהמשפט On error הורה לתוכנית לאן לפנות במקרה וקורית שגיאה. בשורות קוד אלו ההפניה היא אל התווית ErrHandler. **תווית** מיוצגת על ידי שמה ונקודתיים אחריו. לדוגמה,

```
ErrHandler:
```

כאשר המשתמש מקליד את הערך 0 (אפס) לתיבת הטקסט Text2 פקודת ההשמה (intDev = Val(Text1.Text) / Val(Text2.Text)) מבצעת חילוק באפס. חילוק זה אינו חוקי וגורם מצב שגיאה. במקרה רגיל שבו אין טיפול במצבי שגיאה, התוכנית היתה "עפה" עם הודעות שגיאה (תרשים 20.3). בשורות הקוד שבדוגמה התוכנית מונחית לפנות אל התווית ErrHandler במקרה של שגיאה, במקום להפסיק את פעולתה מייד.



תרשים 20.3

בשורות הקוד הבאות לאחר התווית אנו מטפלים באופן ידידותי בשגיאה, ולא עוצרים את התוכנית. בשורות הקוד שבדוגמה לעיל מוצגת הודעה למשתמש המעירה לו כי ביצע פעולה לא חוקית, חילוק באפס, ונותנת לו אפשרות להקליד מספר אחר.

על ידי המשפט On Error ניתן להפנות את התוכנית גם לכל שורה רצויה, במקום להפנות אותה לתווית. צריך לציין את מספר השורה, כך :

```
On Error GoTo 31
...
31 MsgBox "You can't division by zero!"
```

השורה צריכה להיות באותה פונקציה או באותה שיגרה שבה הופעל משפט On Error. במקרה זה, "מספרי השורה" ניתנים על ידי המתכנת ומשמשים למעשה כתוויות.

פתרון שגיאות

בדוגמה הקודמת יצאנו מתוך הנחה שהשגיאה קרתה כתוצאה מחילוק באפס. אך ייתכן שהשגיאה קרתה בגלל סיבה אחרת כלשהי. לדוגמה, המשתמש הקליד בתיבת הטקסט מחרוזת במקום ערך מספרי. במקרה זה, ההודעה שמוצגת אינה מתאימה. איך נדע איזו שגיאה קרתה כאן?

מספר השגיאה

בכל פעם שקורית שגיאה בתוכנית, ויז'ואל בייסיק יוצרת אובייקט מסוג **Err** ששניים ממאפייניו העיקריים הם : Number ו- Description.

Number מייצג את מספר השגיאה שקרתה. לכל שגיאה הקורית במהלך התוכנית יש מספר המייצג אותה. לדוגמה, שגיאה מס' 6, "OverFlow", היא שגיאת גלישה שנגרמת כתוצאה מביצוע פקודת השמה של ערך החורג מתחום קיבולת המשתנה. השגיאה "Division by zero" הנובעת מחילוק באפס, מיוצגת על ידי המספר 11, וכן הלאה.

מעתה, כאשר נכתוב את שורות הקוד לטיפול בשגיאה, נברר תחילה את מספר השגיאה. מספר השגיאה, שהוא ערך המאפיין Number של האובייקט Err, יסייע במתן הודעות מתאימות למשתמש ובמתן פתרון מדויק. לדוגמה:

```
Private Sub cmdDir_Click()
   On Error GoTo ErrHandler
   Dim intDiv As Integer
   intDiv = Val(Text1.Text) / Val(Text2.Text)
   Exit Sub

ErrHandler:
   Select Case Err.Number
     Case 6 :
        MsgBox "OverFlow, please enter appropriate value!"
        Text1.SetFocus
     Case 11:
        MsgBox "Division by zero!"
        Text2.SetFocus
   End Select
End Sub
```

שים לב שלפני התווית ErrHandler תבוא בדרך כלל הפקודה Exit ליציאה מהשיגרה או הפונקציה, מכיון שכאשר לא קורית שגיאה והתוכנית פועלת כשורה, אין צורך וגם לא רצוי להיכנס לקטע הקוד שאחרי התווית ErrHandler.

שם התווית ErrHandler הינו רשות וניתן לקבוע לה כל שם רצוי מלבד מילים שמורות. לדוגמה:

```
On Error GoTo Lbl
....
Lbl:
....
```

תיאור השגיאה

מאפיין חשוב נוסף של האובייקט Err שהזכרנו בסעיף הקודם הוא **Description**. מאפיין זה מציג את תיאור השגיאה. מספר השגיאה אינו אומר דבר למשתמש (מספר השגיאה שימושי יותר למתכנת), אולם הודעה עם תיאור השגיאה יכולה להנחות אותו לתיקון התקלה ולמנוע ממנו חזרה עליה. כדי להציג את תיאור השגיאה, כפי שניתן על ידי ויז'ואל בייסיק, נשתמש במשפט **Err.Description**. לדוגמה:

```
    on Error GoTo ErrHandler
    ...
ErrHandler:
   MsgBox Err.Description
```

אין הכרח להשתמש בתיאור של ויז'ואל בייסיק לשגיאה, ניתן להציג על סמך מספר שגיאה הודעה כלשהי אחרת בעברית.

המשפט Resume

בחן את שורות הקוד הבאות:

```
Private Sub cmdOK_Click()
   On Error GoTo Errhandler
   Dim Num As Integer
   Num = txtNum.Text
   Exit Sub

ErrHandler:
   MsgBox "Enter only numbers!"
End Sub
```

כשהמשתמש לוחץ על לחצן cmdOK הוא מפעיל את האירוע Click שמוצמד לו. באירוע מתבצעת פקודת השמה, המציבה את הערך שבתיבת הטקסט txtNum אל המשתנה Num. כאשר המשתמש מקליד ערך שאינו מספרי, כמו למשל את שמו, פקודת ההשמה שגויה (שגיאה מס׳ 13 - Type Mismatch) מכיון שנעשה כאן ניסיון להכניס מחרוזת למשתנה מספרי.

מכיון ששורות הקוד "מוגנות" על ידי המשפט On Error המטפל בשגיאות, ברגע שהשגיאה קורית, התוכנית אינה מופסקת, אלא פונה אל שורות הקוד העוסקות בדיווח ובתיקון השגיאה. במקרה זה הטיפול בשגיאה הוא הצגת הודעה למשתמש "Enter only numbers". כאשר המשתמש מאשר את ההודעה, התוכנית ממשיכה בשורת הקוד הבאה. בדוגמה שלנו שורת הקוד הבאה היא End Sub; התוכנית עוברת אליה ומסיימת את השיגרה.

לעיתים נרצה לתת למשתמש הזדמנות שנייה לתקן את התקלה מבלי לסיים את הפונקציה או את השיגרה. לדוגמה, כאשר המשתמש מנסה לקרוא קובץ מכונן :A ואין דיסקט בכונן, הפעולה גורמת להודעת שגיאה. נניח שהשגיאה מטופלת כך שלמשתמש מוצגת הודעה שעליו להכניס דיסקט לכונן. במצב זה, יציאה מהשיגרה לא תהיה נכונה, מכיון שלאחר תיקון התקלה והכנסת דיסקט לכונן, התוכנית צריכה לקרוא את הקובץ מהדיסקט, בשעה שיציאה יכולה לפגוע בפעולת התוכנית. הצעד הנכון יהיה אם כן, לחזור אל שורת הקוד שבה קרתה התקלה ולאפשר לתוכנית לבצע אותה שוב, כאשר הדיסקט נמצא בכונן.

כדי להשיג זאת, נשתמש במשפט **Resume**. המשפט Resume מורה לתוכנית לחזור אל השורה שבה קרתה השגיאה לאחר סיום הטיפול בשגיאה. לדוגמה:

```
Private Sub cmdOpen_Click()
   On Error GoTo ErrHandler
   Open txtFile.Text For Input As #1
   ...
   Close #1
   Exit Sub
```

```
ErrHandler:
   If Err.Number = 71 Then
      MsgBox "Enter Diskette to Drive A:"
      Resume
   End IF
End Sub
```

שים לב ששגיאה 71 מציינת כי הכונן שממנו אנו רוצים לקרוא אינו מוכן, או במילים אחרות: אין בו דיסקט. לאחר הודעה מתאימה המשתמש יכול להכניס את הדיסקט ואז התוכנית תנסה לפתוח שוב את הקובץ. בשגרת הטיפול בשגיאה השתמשנו במשפט **Resume** המחזיר את התוכנית לשורת הקוד ...Open שבה מתבצע ניסיון נוסף לפתוח את הקובץ. אם המשתמש לא הכניס את הדיסקט, השגיאה חוזרת ושוב מופיעה ההודעה המורה לו להכניס את הדיסקט. זה יימשך עד אשר המשתמש יבצע את הפעולה הנדרשת. לעיתים מוסיפים מונה לשורות הקוד המטפלות בשגיאה, אשר ימנה את הפעמים אשר קרתה בהם השגיאה. כאשר המונה עובר על ערך נתון, נניח 3, מפסיקים את הפעולה וממשיכים בתוכנית. לדוגמה:

```
Private Sub cmdOpen_Click()
   On Error GoTo ErrHandler
   Dim I As Integer
   Open txtFile.Text For Input As #1
   ...
   Close #1
   Exit Sub

ErrHandler:
    I = I + 1
    If I > 3 Then Exit Sub
    IF Err.Number = 71 then
        MsgBox "Enter Diskette to Drive A:"
        Resume
    End If
End Sub
```

המשפט Resume Next

אפשרות נוספת לטפל בשגיאה היא להורות למחשב להתעלם ממנה ולהמשיך. תחביר המשפט המורה לתוכנית להתעלם מהשגיאה ולהמשיך כרגיל:

```
On Error Resume Next
```

במקום לשלוח את התוכנית אל שגרת הטיפול בשגיאות, מורה לה המשפט Resume Next להמשיך למשפט הבא שלאחר שורת השגיאה.

לדוגמה:

```
Private Sub cmdOpen_Click()
   On Error Resume Next
   Open txtFile.Text For Input As #1
   ...
   Close #1
End Sub
```

אם תקרה שגיאה בניסיון לפתוח את הקובץ, התוכנית תתעלם מהשגיאה ותמשיך לשורת הקוד הבאה. שים לב, אם יימצאו שגיאות נוספות באותה שיגרה, ההתייחסות אליהן תהיה באופן דומה: התעלמות וקפיצה לשורה הבאה.

פונקציה הקוראת לפונקציה

מבנה מודולרי של שורות הקוד כולל גם קריאה לפונקציה מתוך פונקציה, ברמות קינון שונות. נניח שפונקציה א' קוראת בשלב מסוים לפונקציה ב'; פונקציה ב' מבצעת את שורות הקוד שבה וחוזרת לפונקציה א', אל שורת הקוד שאחרי שורת הקריאה. אך מה קורה כאשר יש שגיאה בפונקציה ב'? מי מזהה את השגיאה, פונקציה ב' או פונקציה א'? ומה קורה כאשר פונקציה ב' קוראת לפונקציה ג' והשגיאה קרתה בפונקציה זו?

הכלל במקרים אלו הוא: אם לפונקציה בה קרתה השגיאה יש קוד לטיפול בשגיאות, התוכנית פונה לקוד זה; אם לא קיים קוד מיוחד כזה בפונקציה, התוכנית פונה לקוד המטפל בשגיאות (אם קיים כזה) הנמצא בפונקציה שקראה לפונקציה בה קרתה השגיאה (בפונקציה שברמה הגבוהה יותר). לדוגמה:

```
Sub A()
   On Error GoTo ErrHandler
   Call B
   Open txtFile.Text for Input As #1
   ...
   Close #1
   Exit Sub
Errhandler:
   MsgBox "Subroutine A ErrHandler!"
End Sub

Sub B()
   On Error GoTo ErrHandler
   Open txtFile.Text For output As #2
   ...
   Close #2
   Exit Sub
ErrHandler:
   MsgBox "Subroutine B ErrHandler!"
End Sub
```

מהלך ביצוע תוכנית הדוגמה: ביצוע השיגרה A. מתוך A מתבצעת קריאה לשיגרה B. התוכנית מנסה לבצע את המשפט Open (נצא מתוך הנחה שיש תקלה) ואז היא נתקלת בשגיאה כלשהי. מכיון שלשיגרה B יש תוכנית טיפול בשגיאות, התוכנית תפנה אליה וההודעה שתוצג: "Subroutine B ErrHandler!". אחר כך התוכנית תסיים את השיגרה B ותחזור לשיגרה A. אם בשיגרה A תהיה שגיאה, תוכנית טיפול השגיאות שלה תטפל בשגיאה שנוצרה.

שים לב לשורות הקוד הבאות:

```
Sub A()
   On Error GoTo ErrHandler
   Call B
   Open txtFile.Text For input As #1
   ...
   Close #1
   Exit Sub
ErrHandler:
   MsgBox "Subroutine A ErrHandler!"
End Sub

Sub B()
   Open txtFile.Text For output As #2
   ...
   Close #2
End Sub
```

בדוגמה זו השיגרה B אינה כוללת טיפול בשגיאות. כאשר התוכנית תבצע את השיגרה A היא תקרא מתוכה לשיגרה B. אם וכאשר תקרה שגיאה במהלך השיגרה B, התוכנית תחפש תחילה את מטפל השגיאות בשיגרה שבה קרתה השגיאה. מכיון שלשיגרה B אין תוכנית טיפול בשגיאות, התוכנית הראשית תפנה למטפל השגיאות של השיגרה או הפונקציה שקראה לשיגרה B, בדוגמה שלנו זו השיגרה A. במקרה זה, התוכנית תדלג אל התווית ErrHandler של שיגרה A ותציג את ההודעה: "Subroutine A ErrHandler", למרות שהשגיאה התבצעה בשיגרה B!

שים לב, כלל זה תקף לכל רמת קינון של שגרות או פונקציות. אם אין תוכנית טיפול בשגיאות באף לא אחת מהשגרות (גם לא לראשונה בהיררכיה), התוכנית הראשית תיפסק במקום בו קרתה השגיאה עם הודעה מתאימה.

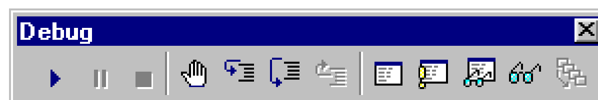
מנפה השגיאות – Debugger

סוג שגיאה שקשה לאיתור כולל את **השגיאה הלוגית**. להבדיל משגיאות תחביר כדוגמת שימוש בפונקציה לא בצורה הנכונה וכו', עליה מתריעה ויז'ואל בייסיק מייד או מאוחר יותר בזמן הידור התוכנית, שגיאה לוגית קשה יותר למעקב. לעיתים אין מבחינים בה, אם מסלול הביצוע אינו עובר בקטע הפגום של התוכנית, או שלא נוצר מצב המחייב ביצוע פעולה שמתברר שהיא שגויה. לדוגמה, חריגה מתחום מערך היא

שגיאה לוגית, ולא ניתקל בה עד אשר המשתמש יכניס ערכים מסוימים שחורגים מגבולות המערך. דוגמה נוספת לשגיאה לוגית יכולה להיות כאשר התוכנית מכניסה לשדה מספרי ערך אלפביתי כלשהו.

שגיאות לוגיות אינן נובעות מתחביר שגוי, מהצהרה שאינה תקינה, או אי הצהרה על משתנה או פונקציה, ולכן הן קשות יותר למעקב. כדי לפתור קושי זה יש כלי תוכנה הקרויים **Debuggers** - מנפי שגיאות - המסייעים במעקב אחר ביצוע שורות הקוד בתוכנית ובמציאת השגיאה הלוגית.

לכלים אלה ניגשים מתפריט **Debug** שבשורת התפריטים. סרגל הכלים **Debug** מכיל גם כן כלים אלה, וניתן לצרפו לסרגל הכלים הסטנדרטי של ויז'ואל בייסיק (תרשים 20.4). להוספת סרגל Debug לחץ לחיצה ימנית על סרגל הכלים וסמן את האפשרות.



תרשים 20.4

נקודת עצירה (Toggle Breakpoint)

לחיצה על האפשרות **Start** בתפריט **Run** (או הקשה על F5 במקלדת) מפעילה את היישום.

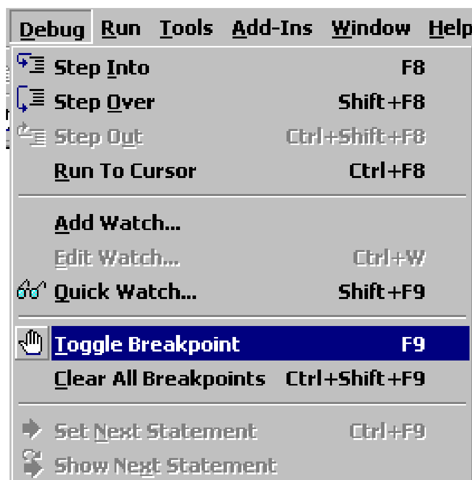
כאשר מבצעים תוכנית, מבצעים למעשה את שורות הקוד זו אחר זו על פי האירועים השונים המתרחשים ביישום (כלחיצה על לחצן, למשל). שורות קוד אלו כוללות שימוש במשתנים, ביצוע פקודות השמה ופעולות אחרות. ערכי המשתנים משתנים בהתאם לפעילות התוכנית, לפי נתונים חיצוניים המוקלדים על ידי המשתמש ועוד.

כאשר הערכים שמקבלים המשתנים נכונים, הכל טוב ויפה; אך מה קורה כאשר משתנה מכיל ערך כלשהו שאינו חוקי מבחינת התנאים, או ערך חריג? במצב זה אנו רוצים לצפות בערך המשתנה באותה נקודה, או במספר נקודות במהלך התוכנית, כדי לקבל מידע על תהליך שינוי הערכים של המשתנה.

כלי העזר של ויז'ואל בייסיק הוא **נקודת העצירה** (**Toggle Breakpoint**). נקודת העצירה מורה לתוכנית לעצור את מהלך ביצוע התוכנית בשורה שבה הוצבה. אך בשונה מהפסקת התוכנית באופן ידני בנקודה מסוימת, הפסקת התוכנית על ידי נקודת העצירה הממוחשבת אינה הפסקה מוחלטת. במצב זה המשתנים קיימים בזיכרון עם הערכים האמיתיים שלהם באותה נקודת זמן. במצב זה ניתן לצפות בערכי המשתנים ולקבל מושג על פעולת התוכנית. לעומת זאת, בהפסקה מוחלטת של התוכנית, כל המשתנים משתחררים מהזיכרון ואין אפשרות לצפות בערכם.

יתרון נוסף בהצבת נקודת עצירה הוא היכולת לצפות בתוצאות ביצוע היישום. בכך נעסוק בהמשך.

כדי להציב נקודת עצירה בשורה מסוימת, הצב את הסמן בשורה זו ובחר באפשרות **Toggle Breakpoint** שבתפריט **Debug**, או הקש על F9 (תרשים 20.5). ניתן להציב מספר נקודות עצירה במהלך התוכנית, ככל שיידרש. עם זאת, רצוי לתכנן את העצירות האלו במצב שניתן לקבל בו מידע רלוונטי.

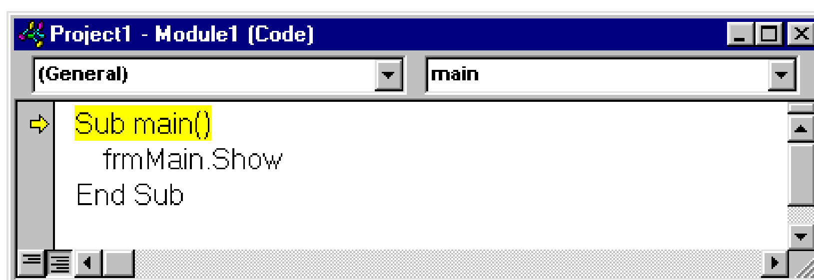


תרשים 20.5

צעד-אחר-צעד (Step Into)

אפשרות נוספת למעקב אחר מסלול ביצוע התוכנית, הוא לצעוד במסלול זה צעד-אחר-צעד, ולמעשה שורה אחר שורה, תוך אפשרות לצפות בערכי המשתנים השונים, מאפייני פקדים וכל ערך שקיים בזיכרון. לדוגמה הפקד txtFile.Text – ניתן לצפות בערך של המאפיין Text של הפקד, גם כאשר אין רואים את הפקד בפועל.

כדי לבצע את התוכנית צעד-אחר-צעד, בחר באפשרות **Step Into** שבתפריט **Debug** או הקש על F8. התוכנית פונה לשורת הקוד הראשונה וצובעת אותה בצהוב ומסמנת חץ משמאל (תרשים 20.6, אך ללא צבע). כדי להתקדם לשורה הבאה הקש F8 והשורה הבאה תיצבע בצהוב, וכן הלאה שורה אחר שורה עד סוף התוכנית.



תרשים 20.6

שים לב שניתן בכל עת לחזור לביצוע סדיר ושוטף של התוכנית בהקשה על F5 (Start).

דילוג על פונקציה (Step Over)

כאשר התוכנית מבצעת את שורות הקוד צעד-אחר-צעד ומגיעה לשורה שבה מתבצעת קריאה לפונקציה, היא נכנסת אל הפונקציה ומבצעת את שורות הקוד שבה צעד-אחר-צעד. אפשר לדלג על הכניסה לפונקציה. בחר באפשרות **Step Over** מתפריט **Debug**, או הקש **Shift+F8** לדילוג על הפונקציה ולמעבר לשורה שאחריה. שים לב שדילוג על הפונקציה **מבצע** את שורות הקוד בפונקציה עצמה, אך לא צעד-אחר-צעד. הסיבה לכך בדרך כלל שאין לנו עניין לבדוק קטע קוד סטנדרטי.

אם מקישים **F8** ונכנסים לפונקציה, ניתן לצאת מתוכה, ולא להמשיך בה צעד-אחר-צעד. נבחר באפשרות **Step Out** שבתפריט **Debug** או בהקשה על **Ctrl+Shift+F8**.

במסגרת צעד-אחר-צעד ניתן להריץ קטע קוד מסוים ברצף ולחזור לביצוע צעד-אחר-צעד. כאשר התוכנית נמצאת בשורה כלשהי ורוצים להמשיך ברצף ואחר כך להמשיך צעד-אחר-צעד, ננהג כך: נביא את הסמן אל השורה שבה נרצה להמשיך צעד-אחר-צעד ונבחר באפשרות **Run To Cursor** מתפריט **Debug** או נקיש **Ctrl+F8**.

הרצת התוכנית מנקודה מסוימת

לפעמים תרצה במהלך ביצוע צעד-אחר-צעד **לדלג** על קטע קוד מסוים. שים לב, לא הרצה מהירה (Ctrl+F8) אלא דילוג ממש, שפירושו: התוכנית לא תבצע את הפקודות הכלולות בו. לצורך זה נציב את הסמן בשורה שאליה נרצה לדלג, ונקיש **Ctrl+F9**; אפשר לבחור באפשרות **Set Next Statement** מתפריט **Debug**. התוכנית מדלגת על השורות עד לשורה המבוקשת ומתעלמת מהן, כאילו לא היו חלק מהתוכנית. לדוגמה:

```
Dim I As Integer
I = 7
I = 3
Print I
```

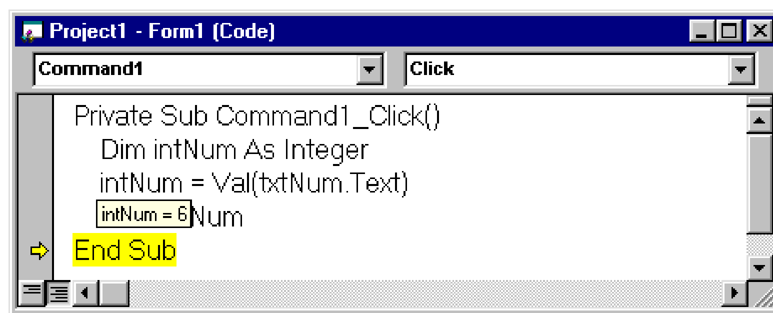
הבה נתרגל את שלמדנו: כשהתוכנית תבצע שורות קוד אלו צעד-אחר-צעד ותבצע את פקודת ההשמה I = 7, ערך המשתנה I יהיה 7. במצב זה, התוכנית נמצאת בשורה I = 3. כעת, נדלג (Ctrl+F9) על השורה I = 3 ונביא את התוכנית אל השורה Print I. מכיון שבוצע דילוג על השורה I = 3, ערכו הנוכחי של I נשמר והתוכנית תדפיס את הערך 7 (מכיון שהיא לא התייחסה לפקודת ההשמה I = 3). שים לב, הרצה בעזרת Ctrl+F8 תגרום לתוכנית להתייחס לשורת הקוד I = 3 והערך שיודפס יהיה 3.

צפייה בערכו של משתנה (Quick Watch)

כל הטכניקות שלמדנו, החל מנקודת עצירה, צעידה בשורות הקוד צעד-אחר-צעד מכוונות לאפשר לנו לצפות בערכי המשתנים ובמאפייני הפקדים לצורך מעקב. ברגע שהתוכנית נעצרה בשורה מסוימת (בנוהל צעד-אחר-צעד) ניתן לצפות בערכי המשתנים השונים שבשורות הקוד.

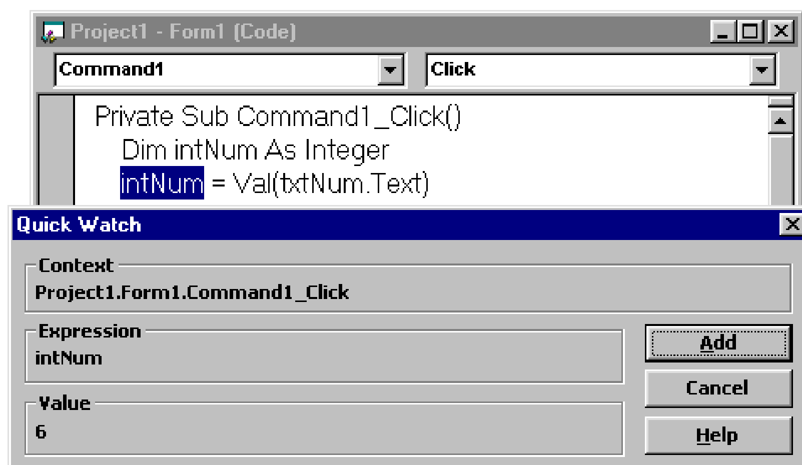
יש שתי דרכים לצפות בערך המשתנה:

1. הבא את סמן העכבר והצב אותו על המשתנה בשורת הקוד (אין צורך ללחוץ עליו). אחרי שנייה או שתיים תופיע תיבת תיאור קטנה (בדומה ל- Tool Text) המציגה את ערך המשתנה (בתרשים 20.7).



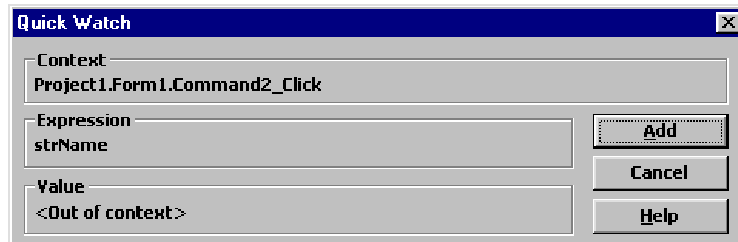
תרשים 20.7

2. בחר במשתנה, או בביטוי, שבערכו ברצונך לצפות על ידי סימון בגרירת העכבר עליו, והקש **Shift+F9**, או בחר באפשרות **Quick Watch** מתפריט **Debug**. החלון Quick Watch ייפתח ובמסגרת Value יופיע ערך המשתנה או הביטוי הנמצא במסגרת Expression (תרשים 20.8).



תרשים 20.8

שים לב שניתן לצפות אך ורק בערכי המשתנים של **אותה** שיגרה או פונקציה, אלא אם עוסקים במשתנים מודולריים או גלובליים שטווח ההכרה שלהם הוא בכל התוכנית. ניסיון לצפות בערך משתנה מקומי של פונקציה או של שיגרה אחרת, יציג את הערך **Out of context** בחלון Quick Watch (תרשים 20.9).

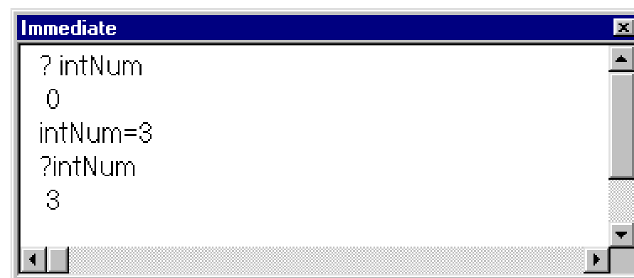


תרשים 20.9

חלון מיידי – Immediate Window

לעיתים הצפייה בערכי משתנה לצורך מעקב אינה מספקת. היה מועיל יותר, לו היינו יכולים לראות מה קורה אילו היה ערכו של המשתנה ערך כלשהו. כלומר, כיצד תנהג התוכנית, או שורת הקוד בה אנו עומדים, במצב כפוי כלשהו.

בוויז'ואל בייסיק קיים חלון Immediate Window אשר נגיש בזמן ריצה בלבד. חלון זה הוא למעשה Editor פשוט שבו ניתן לבצע פקודות ומשפטים בוויז'ואל בייסיק על משתנים בתוכנית ולקבל תשובה מיידית. אם ערכו של המשתנה intNum היה 2 ובחלון Immediate Window נשנה בעזרת פקודת השמה את ערכו ל- 3 (תרשים 20.10), שורות הקוד בתוכנית יתייחסו למשתנה intNum כבעל ערך 3. השפעת ביצוע פקודת ההשמה שלנו תהיה מיידית. שים לב, התו **?** מקביל לפקודה Print בוויז'ואל בייסיק. על הפקודות המבוצעות בחלון Immediate Window לעמוד בחוקי תחביר של השפה.



תרשים 20.10

באופן זה נוכל לבצע שינוי בכל משתנה או ביטוי של השיגרה או הפונקציה שבה נמצאת התוכנית בשלב זה, וגם לראות כיצד מגיבות שורות הקוד לשינוי.

כדי להציג את החלון Immediate Window הקש **Ctrl+G**.

תרגול

1. בנה תוכנית המאפשרת למשתמש לקרוא קובץ מהדיסקט. טפל בכל השגיאות שאתה צופה שעלולות להיות, כמו לדוגמה, טיפול במקרה שבכונן אין דיסקט.

2. בנה תוכנית המאפשרת למשתמש להכניס שני מספרים ולבצע פעולה חשבונית עליהן. כתוב מטפל בשגיאות אשר יטפל בכל השגיאות העלולות לקרות. לדוגמה, חילוק באפס, הכנסת תו במקום מספר וכו'.

21:
הכנת ערכת התקנה
(Setup kit)

סיימנו את הפרויקט, בדקנו אותו וניפינו שגיאות. כל שנשאר הוא למסור את היישום ללקוח כדי שיוכל להשתמש בו כפי שתוכנן. לשם כך עלינו לבצע שתי פעולות עיקריות:

1. יצירת קובץ הפעלה (EXE).
2. הכנת ערכת התקנה.

ללא שני שלבים אלה המשתמש לא יוכל להפעיל את היישום במחשב שברשותו, אלא אם קיימת בו סביבת הפעלה או סביבת פיתוח של ויז'ואל בייסיק. מכיון שאין אנו רוצים להתקין את תוכנת ויז'ואל בייסיק (הכוללת את סביבת הפיתוח) אצל כל לקוח שמשתמש ביישום, חובה עלינו למסור למשתמש קובץ הפעלה (EXE) וגם ערכת התקנה עבור היישום.

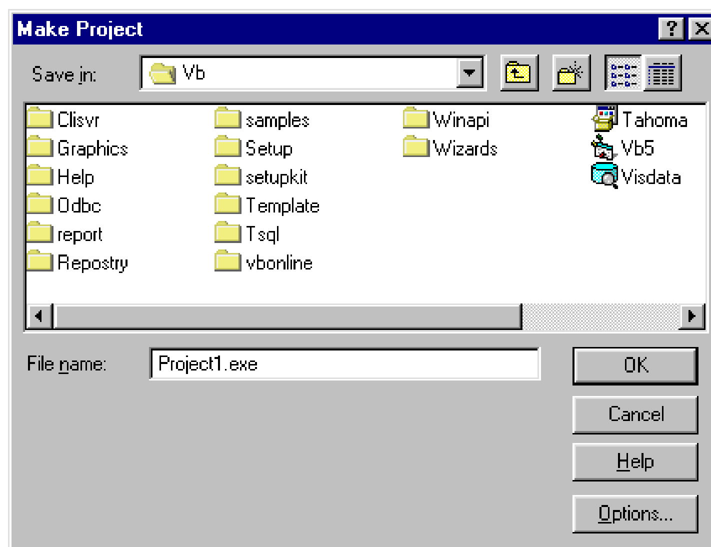
ומדוע אי אפשר להסתפק בקובץ EXE? למרות שקובץ EXE הוא קובץ הפעלה, הוא נעזר בקבצי DLL שונים. במקום לשמור את כל המידע בקובץ EXE אחד גדול, הוא מחולק, כמקובל ביישומים חלונאיים רבים, למספר קבצי DLL. קבצים אלה אינם חלק ממערכת ההפעלה Windows וחייבים להתקין אותם במיוחד בספריות המתאימות במערכת.

קובץ ההפעלה בנוי לא רק מקבצי DLL, אלא גם מקבצים נוספים כקבצי OCX למיניהם וקבצים אחרים כקבצי MDB (של מסד הנתונים Microsoft Access) ועוד. בכל מקרה, קובץ ההפעלה EXE אינו יכול לפעול כקובץ עצמאי ולכן ערכת ההתקנה חיונית להפעלתו במחשב של המשתמש.

יצירת קובץ EXE

יצירת קובץ EXE פשוטה מאוד. צריך לבחור באפשרות **Make Project1.exe** שבתפריט **File**. כאשר לפרויקט יש שם, הוא יופיע במקום המילה Project1, כמו לדוגמה Make MyApp.exe לפרויקט בשם MyApp.

בחירה באפשרות זו גורמת לפתיחת חלון בו נבחר בתיקיה שבה תיצור ויז׳ואל בייסיק את קובץ ההפעלה ונציין את שם הקובץ (תרשים 21.1).



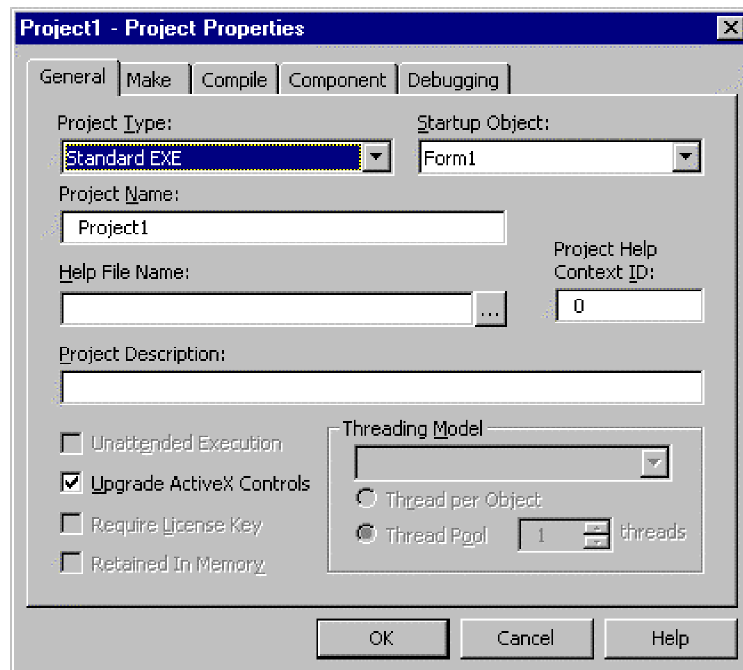
תרשים 21.1

שם הקובץ ושם הפרויקט לא חייבים להיות זהים. לחיצה על OK לאישור תחל את פעולת ההידור ויצירת קובץ EXE. בסוף התהליך יופיע בתיקיה הרצויה קובץ EXE שנושא את השם שנקבע לו.

ערכת ההתקנה יוצרת באופן אוטומטי קובץ EXE גם אם לא יצרנו אותו באופן ידני קודם לכן. לפעמים נרצה ליצור קובץ EXE ללא ערכת התקנה, כמו לדוגמה, כאשר התקנו את היישום פעם אחת אצל המשתמש ואנו מוסרים לו גירסה חדשה שלו. במקרה זה אפשר להסתפק ביצירת קובץ EXE ומסירתו ללקוח.

הגדרות הפרויקט

לפני יצירת קובץ ההפעלה ניתן לקבוע הגדרות ומאפיינים שונים לפרויקט. בחר באפשרות **Project1 Properties** שבתפריט **Project** (אם יש שם לפרויקט הוא יופיע במקום "Project1"). בחירה באפשרות זו מציגה את החלון **Project Properties** (תרשים 21.2).



תרשים 21.2

בחלון מוצגות מספר כרטיסיות. הראשונה שבהן היא **General**, שבה מעדכנים את המאפיינים הכלליים של הפרויקט, שהחשובים בהם :

1. **Project Name** - שם הפרויקט.
2. **Help File Name** - אם ליישום יש קובץ עזרה (HLP) נקליד בתיבה זו את שם קובץ העזרה ואת הנתיב. לימין תיבת הטקסט נמצא הלחצן ... המאפשר דפדוף במערכת ההפעלה וחיפוש הקובץ הרצוי.
3. **Project Description** - תיאור קצר על הפרויקט.

בכרטיסיה **Make** נקבעים המאפיינים הדרושים לוויז'ואל בייסיק בזמן ההידור ויצירת קובץ EXE (תרשים 21.3). המאפיינים העיקריים הם :

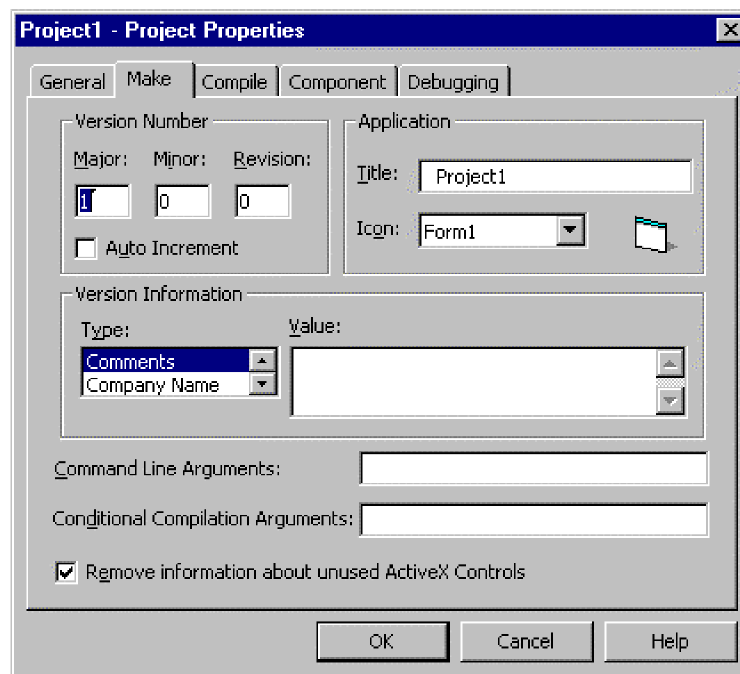
1. **Version Number** - גרסת היישום. ניתן לשנות את הגירסה באופן ידני או לפנות לוויז'ואל בייסיק שתדאג לשינוי הגירסה באופן אוטומטי בכל פעם שנוצר קובץ EXE חדש.

מספר גירסה זה נמצא תחת הכותרת **File version** (תרשים 21.4), כחלק ממאפייני קובץ EXE שכלולים באפשרות "מאפיינים" שבתפריט המקוצר (כאשר לוחצים לחיצה ימנית על שם הקובץ בסייר).

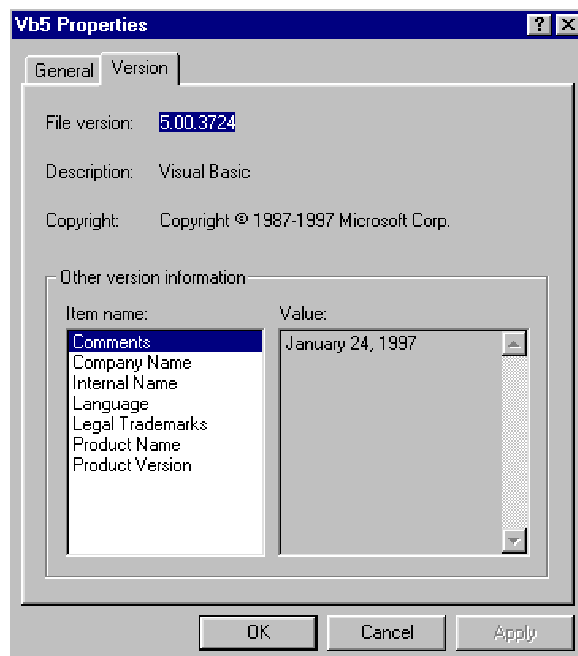
2. **Application** - בחלק זה נקבעת כותרת הפרויקט והסמל שלו. סמל זה יהיה גם סמל קובץ ההפעלה (EXE).

3. **Version Information** - מידע כללי לגבי היישום, אשר יוצג כחלק ממאפייני קובץ ההפעלה.

בתיבת הרשימה **Type** בוחרים במאפיין שנרצה לקבוע, ובתיבת הטקסט **Value** מקלידים את ערכו.



תרשים 21.3



תרשים 21.4

בכרטיסיה **Compile** נקבעים המאפיינים הקשורים לתהליך הידור הפרויקט. כאן ניתן לקבוע תנאים להידור.

תוכניות ההתקנה Package & Deployment Wizard

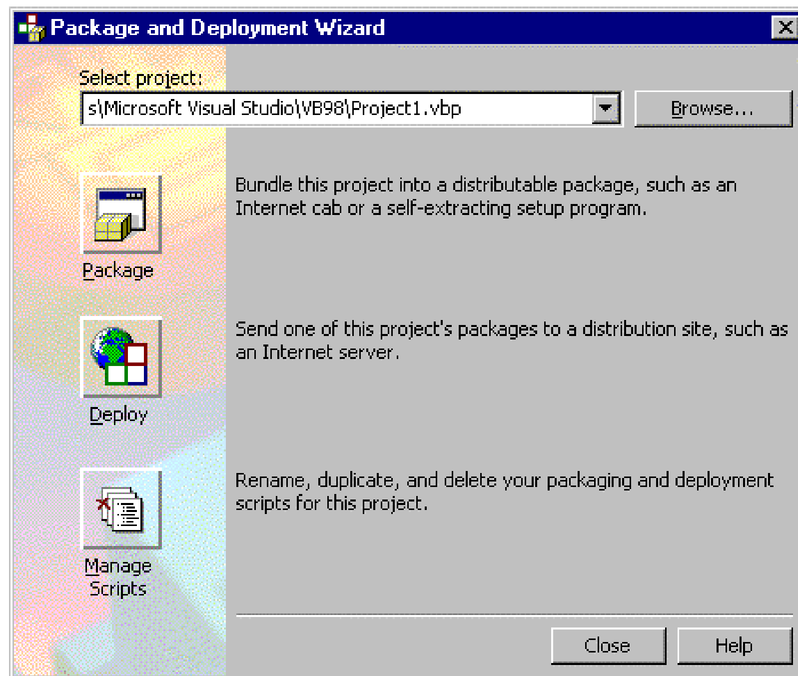
כדי לאפשר ליישום לפעול גם במחשב המשתמש, נדרשת התקנת סביבה מתאימה במחשב. סביבה זו כוללת את כל הקבצים עליהם "בנוי" קובץ ההפעלה כמו DLL, OCX ואחרים. אין צורך לדעת בדיוק איזה קובץ דרוש לפרויקט ואיזה לא. ויז'ואל בייסיק מספקת **תוכנית התקנה** - **Package & Deployment Wizard** המכינה בהנחייתנו את ערכת ההתקנה. התוכנית דואגת לבחור את כל הקבצים הדרושים להפעלה תקינה.

ערכת ההתקנה אינה יודעת לכלול קבצים שאינם חלק מסביבת העבודה של ויז'ואל בייסיק, כדוגמת קובץ MDB (קובץ Access) שאותו מנהל היישום. במקרה זה, עלינו להגדיר במפורש לערכת ההתקנה לכלול בה קובץ זה.

תוכנית השירות **Package & Deployment Wizard** נמצאת בקבוצת היישומים Microsoft Visual Studio 6 Tools.

שלב א'

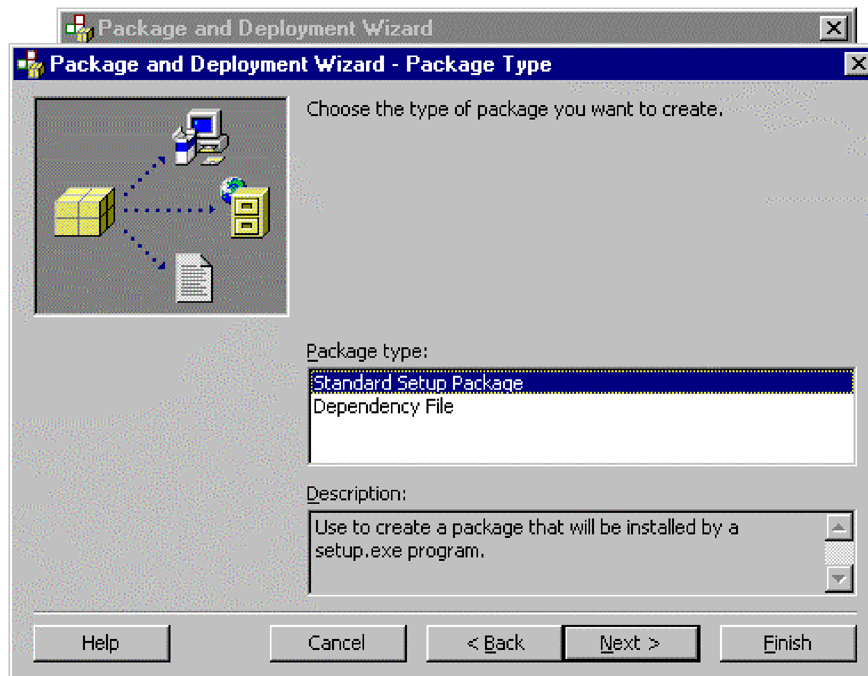
בשלב ראשון יש לבחור את הפרויקט שעל בסיסו נבנה את תוכנית ההתקנה. בתיבת הטקסט **Select Project** נרשם שם קובץ הפרויקט (VBP) והנתיב עבורו נבנה את תוכנית ההתקנה. היעזר בלחצן **Browse** כדי למצוא את מיקומו המדויק של הפרויקט (תרשים 21.5). בחר באפשרות **Package** כדי לעבור לשלב הבא.



תרשים 21.5

שלב ב'

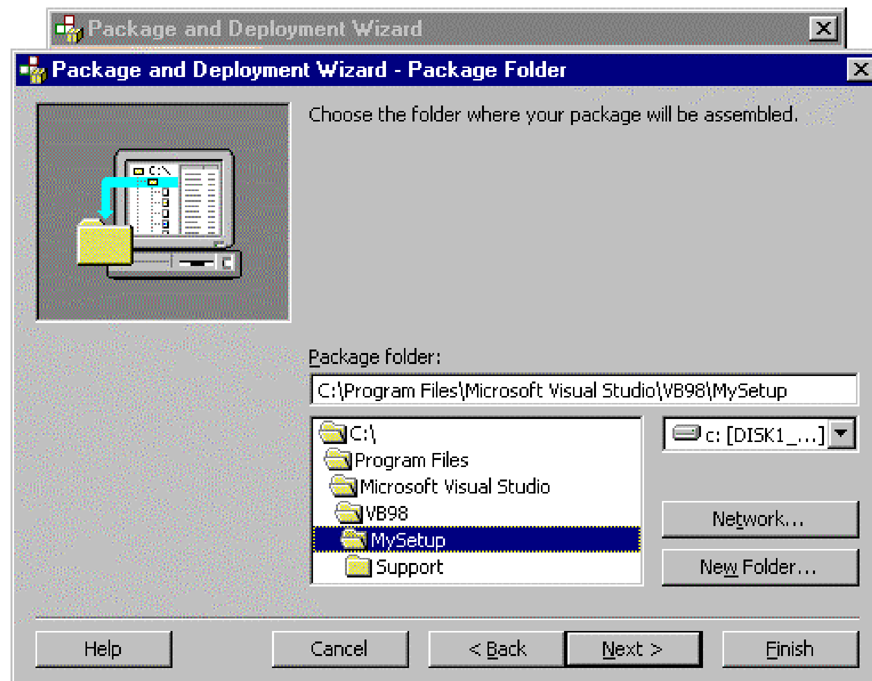
בשלב זה נגדיר לאשף איזה סוג של ערכה עליו לבנות (תרשים 21.6). באפשרותך לבחור בין בניית ערכת התקנה סטנדרטית (**Standard Setup Package**) ובין יצירת קובץ dep (**Dependency File**). קובץ זה מכיל את רשימת כל הקבצים בהם תלוי קובץ ההרצה (EXE) שנוצר מהפרויקט שלך. בלי קבצים אלה, התוכנית שלך לא תפעל במחשב הלקוח (אלא אם קבצים אלו קיימים אצלו במחשב).



תרשים 21.6

שלב ג'

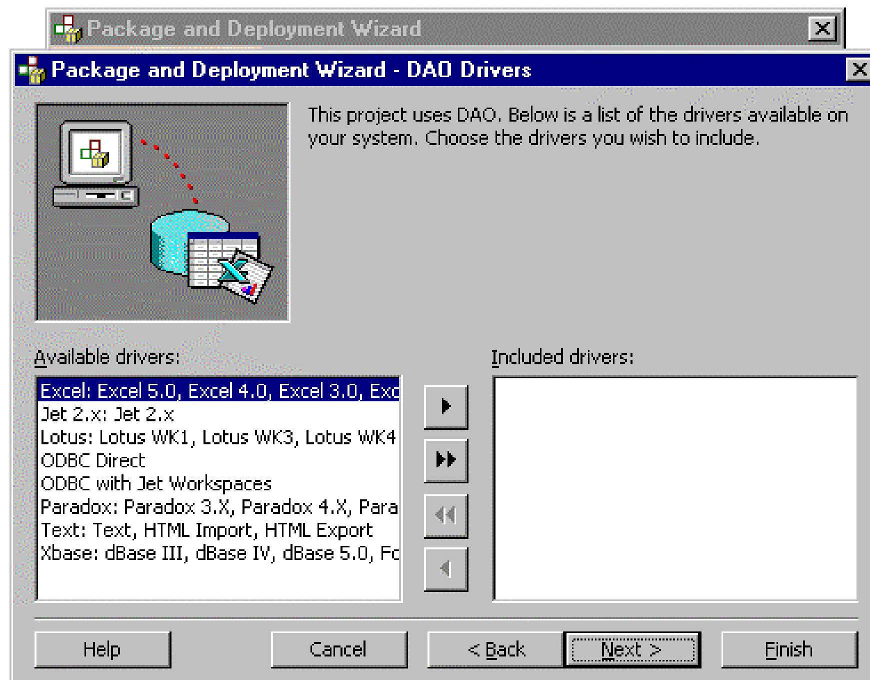
בשלב זה בחלון זה יש לקבוע את תיקיית היעד עבור הקבצים המכווצים של ערכת ההתקנה (תרשים 21.7). ניתן לרשום שם של תיקיה שאינה קיימת במערכת. תוכנת ההתקנה תשאל אם רוצים שהיא תיצור אותה, ובאישור חיובי היא תעשה זאת.



תרשים 21.7

שלב ד'

בשלב זה מופיע החלון **DAO Drivers** (תרשים 21.8).



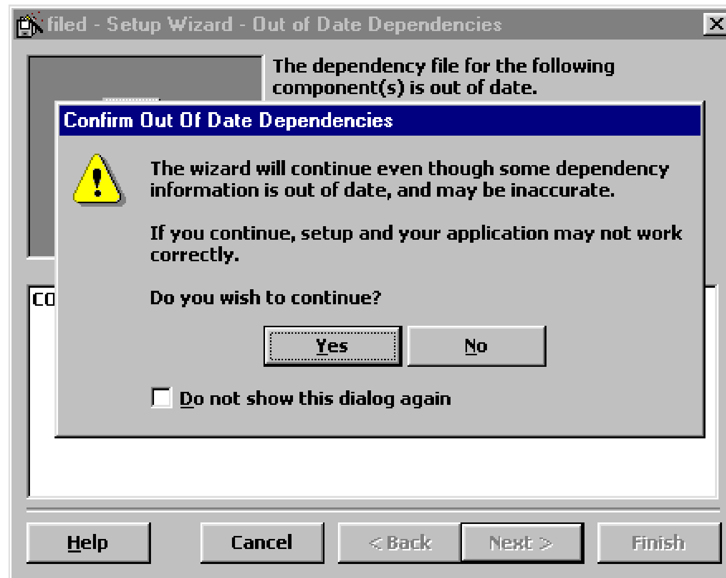
תרשים 21.8

אשף ההתקנה גילה כי בפרויקט נעשה שימוש באובייקט DAO (לצורך ניהול מסד נתונים). במקרה זה מוצגת לפנינו רשימה של מנועים, בעזרתם ינהל האובייקט את מסד הנתונים, מהם נוכל לבחור.

שים לב, שלב זה מופיע אך ורק כאשר תוכנית ההתקנה מזהה שנעשה שימוש בפרויקט באחד מהאובייקטים או מהפקדים של מסד הנתונים, כדוגמת DAO (Date Access Object).

שלב ה׳

בשלב זה מופיע החלון **Out Of Date Dependencies** (תרשים 21.9).



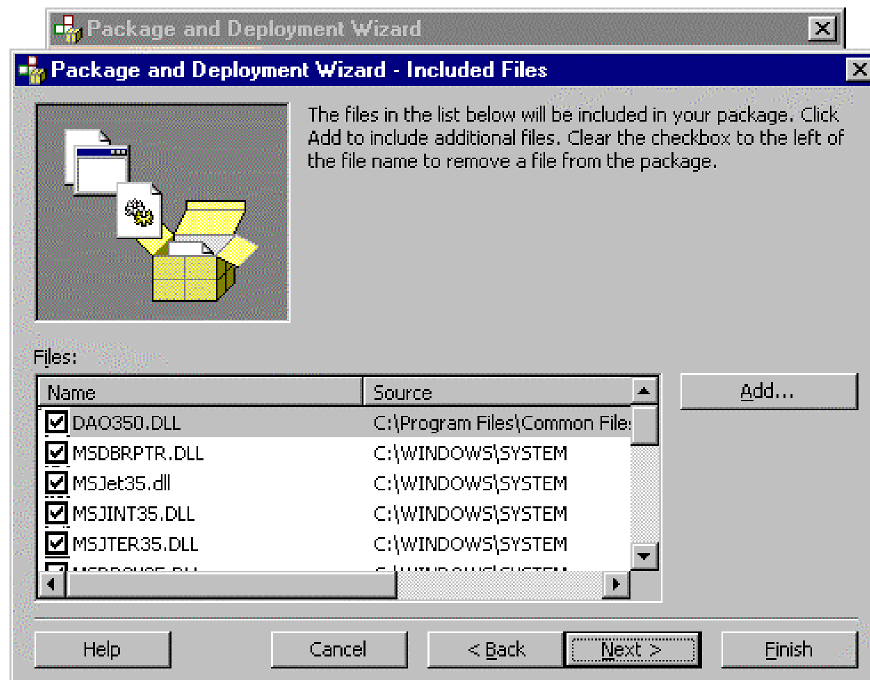
תרשים 21.9

חלון זה אינו מופיע תמיד. אולם אם תוכנת ההתקנה מגלה שאחד או יותר מהקבצים הנדרשים לערכת ההתקנה שנעשה בהם שימוש בפרויקט אינם מעודכנים או מגירסה קודמת, היא מעירה על כך ויש להפסיק את בניית ערכת ההתקנה.

ניתן להמשיך (לחצן "Next>"), אולם במקרה זה התוכנית תזהיר שבמידה ונמשיך עלולים להיות שיבושים בניסיון ההפעלה של היישום (תרשים 21.9). ניתן להתעלם מהודעה זו ולהמשיך לשלב הבא.

שלב ו'

בשלב זה מופיע החלון **Included Files** (תרשים 21.10).

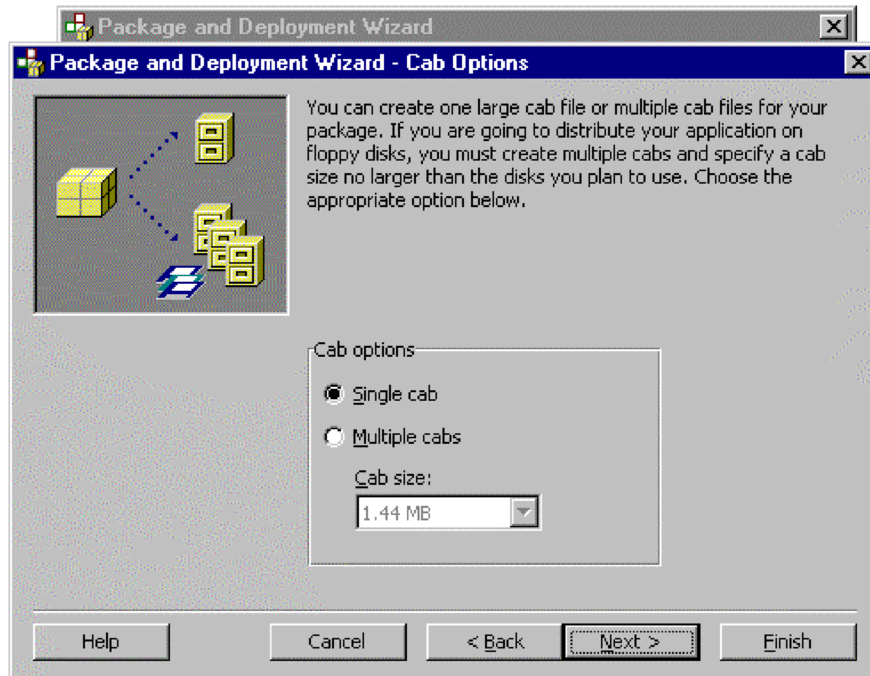


תרשים 21.10

שלב זה קיים אך ורק כשנעשה שימוש בפקד מיוחד בפרויקט. כאשר הוספנו פקד ActiveX Control לארגז הכלים בעת פיתוח הפרויקט, תוכנת ההתקנה מזהה אותו ברשימה. באופן אוטומטי כל תיבות הסימון מסומנות וכוללות את כל הקבצים בערכת ההתקנה. עם זאת, ניתן לבטל את הסימון מתיבת הסימון שמשמאל לשם הקובץ המייצג את הפקד, ולא לכלול אותו בערכת ההתקנה. לדוגמה, כאשר ידוע מהתקנות קודמות שקבצים אלה כבר מותקנים במחשב של המשתמש אין צורך לכלול אותם שוב בערכת ההתקנה הנוכחית.

שלב ז'

בשלב זה מופיע החלון **Cab Options** (תרשים 21.11).



תרשים 21.11

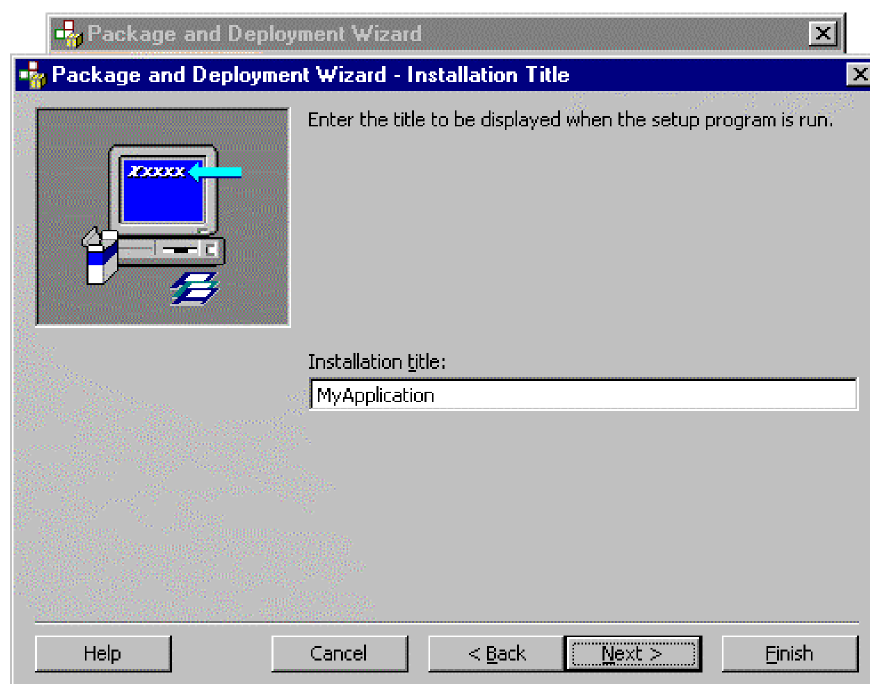
בשלב זה בוחרים את הדרך בה נרצה להפיץ אחר כך את ערכת ההתקנה. שתי האפשרויות הן:

1. **Single Cab** (או Single Directory בגירסה 5) - קובץ גדול אחד (או ספריה אחת בגירסה 5). במקרה כזה כל הקבצים ימוקמו בקובץ אחד בעל סיומת CAB.
2. **Multiple Cabs** (או Floppy Disk בגירסה 5) - דיסקטים. במקרה זה ערכת ההתקנה תחלק את הקבצים למספר קבצי CAB (או למספר דיסקטים בגירסה 5), על פי גודל הפרויקט.

בחר באפשרות הרצויה ועבור לשלב הבא.

שלב ח'

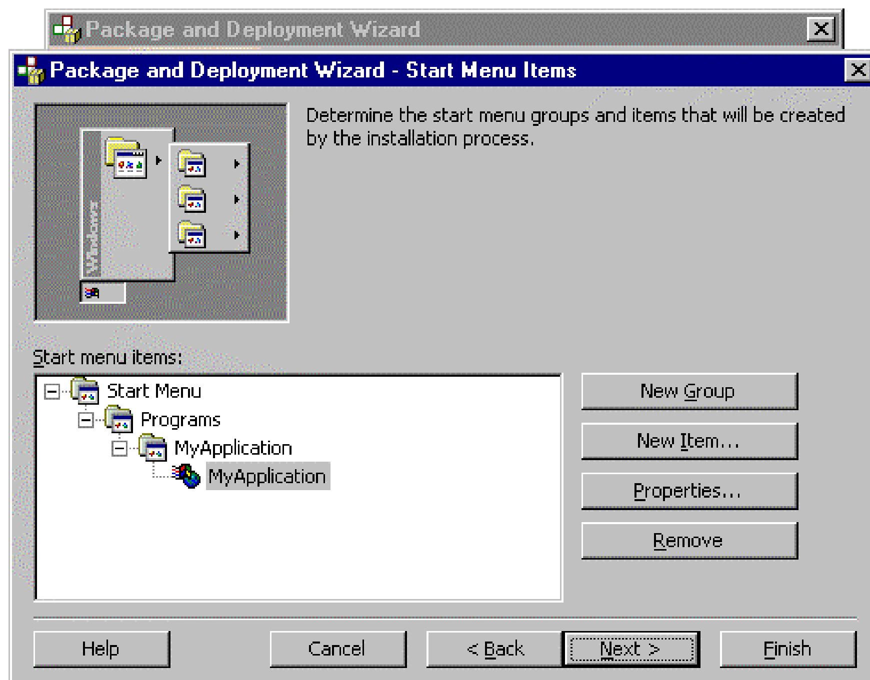
בשלב זה קובעים שם לערכה (תרשים 21.12).



תרשים 21.12

שלב ט׳

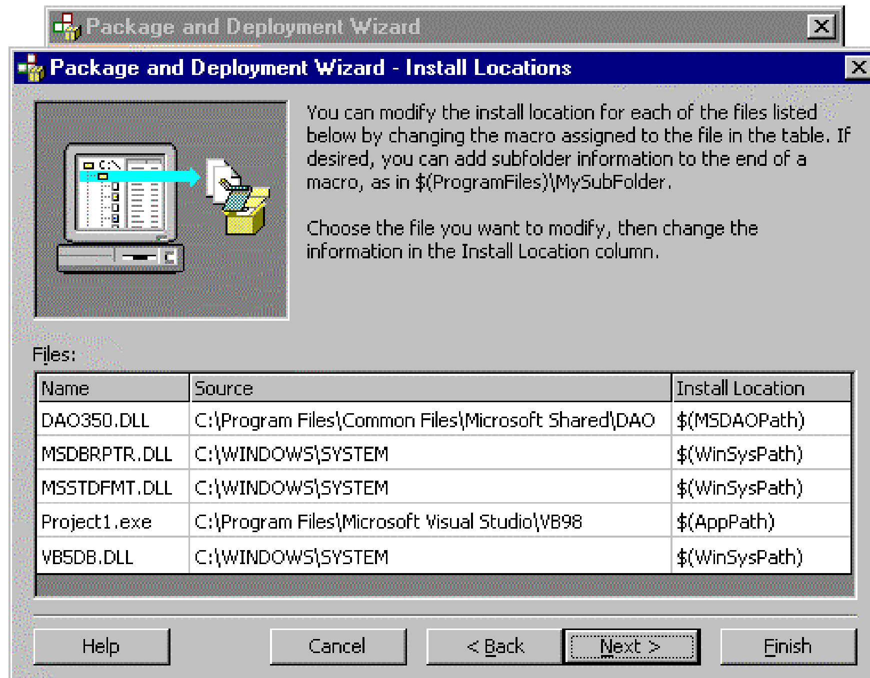
בשלב זה קובעים היכן הערכה תתקין את היישום בתפריט **התחל** של Windows (תרשים 21.13).



תרשים 21.13

שלב י׳

בשלב זה מופיע החלון **Install Locations** (תרשים 21.14).



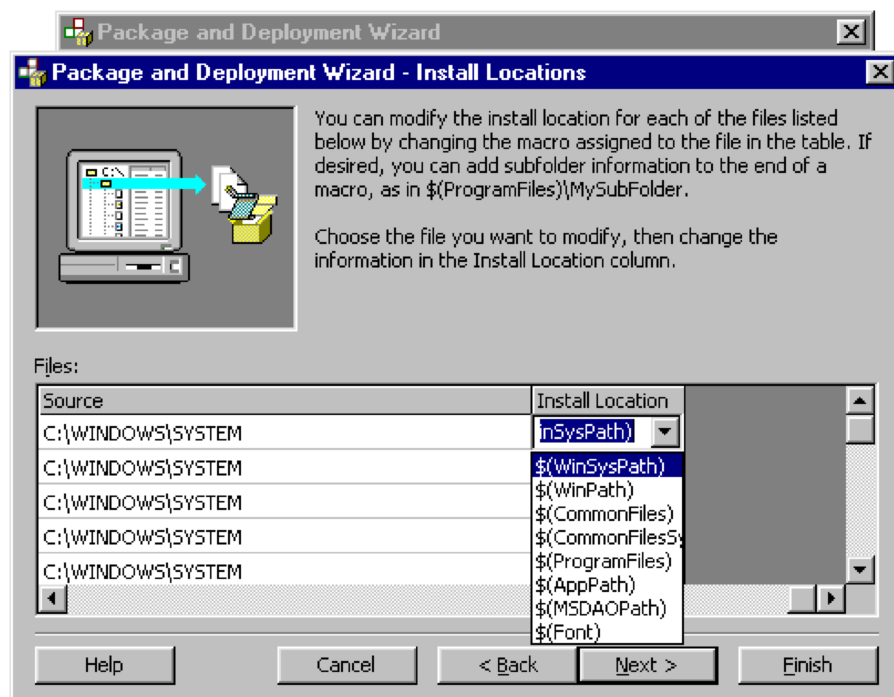
תרשים 21.14

ערכת ההתקנה פועלת בנוהל זה:

את קובץ ההפעלה ואת הקבצים הנוספים שצורפו לרשימה באופן ידני היא מתקינה בתיקיית היישום. את קבצי DLL ואת קבצי OCX היא מתקינה בתיקיה System שנמצאת בתיקיית מערכת ההפעלה.

זו אמנם ברירת המחדל, אך ניתן לשנות לכל קובץ וקובץ את יעד התיקיה שבה יותקן. לדוגמה, נשנה את תיקיית היעד של קבצי DLL לתיקיית היישום.

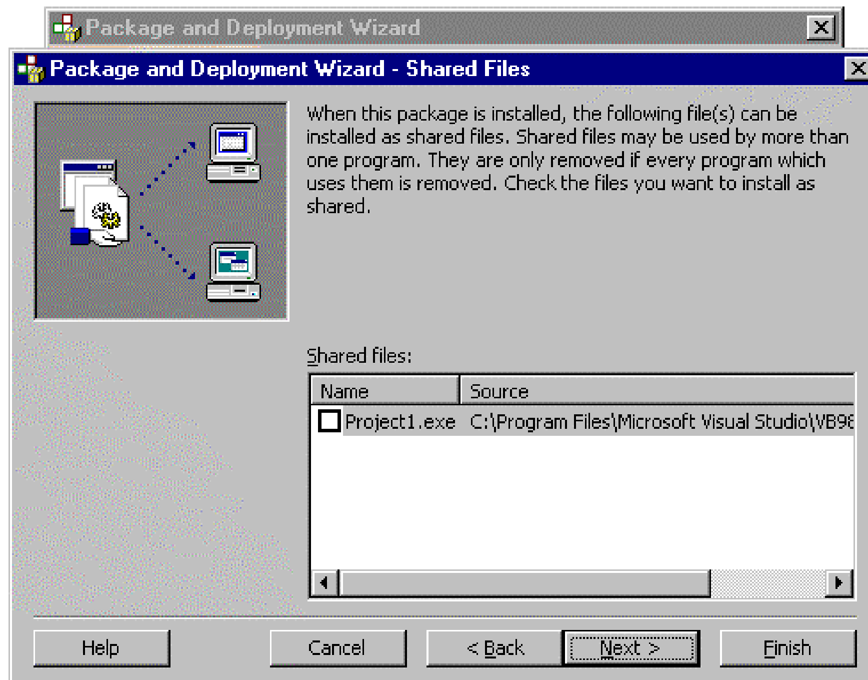
לביצוע השינוי נבחר בעמודה **Install Location** של הקובץ שנרצה לשנות. התא בטבלה ייהפך לתיבה נפתחת (תרשים 21.15). מתיבה זו נבחר את המיקום הרצוי בו תתקין תוכנית ההתקנה את הקובץ שנבחר.


Package and Deployment Wizard
Package and Deployment Wizard - Install Locations
You can modify the install location for each of the files listed below by changing the macro assigned to the file in the table. If desired, you can add subfolder information to the end of a macro, as in $(ProgramFiles)\MySubFolder.
Choose the file you want to modify, then change the information in the Install Location column.
Files:
Source
Install Location
C:\WINDOWS\SYSTEM
C:\WINDOWS\SYSTEM
C:\WINDOWS\SYSTEM
C:\WINDOWS\SYSTEM
C:\WINDOWS\SYSTEM
nSysPath)
$(WinSysPath)
$(WinPath)
$(CommonFiles)
$(ProgramFiles)
$(AppPath)
$(MSDAOPath)
$(Font)
Help
Cancel
< Back
Next >
Finish

תרשים 21.15

שלב י"א

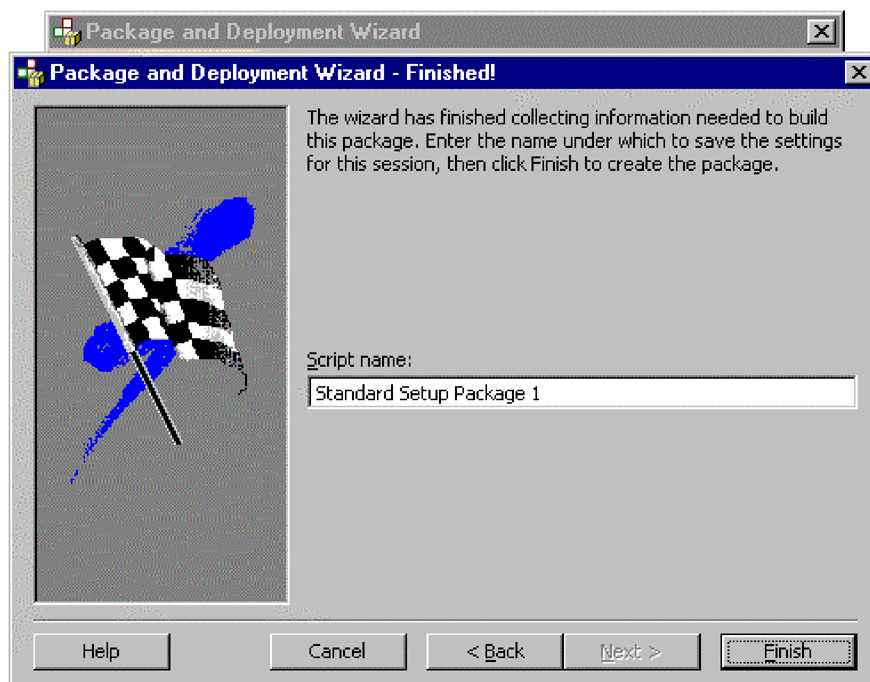
בשלב זה ניתן לקבוע אם קובץ היישום יהיה משותף או לא. התרשים הבא מציג את החלון של שלב זה:



תרשים 21.16

שלב י"ב

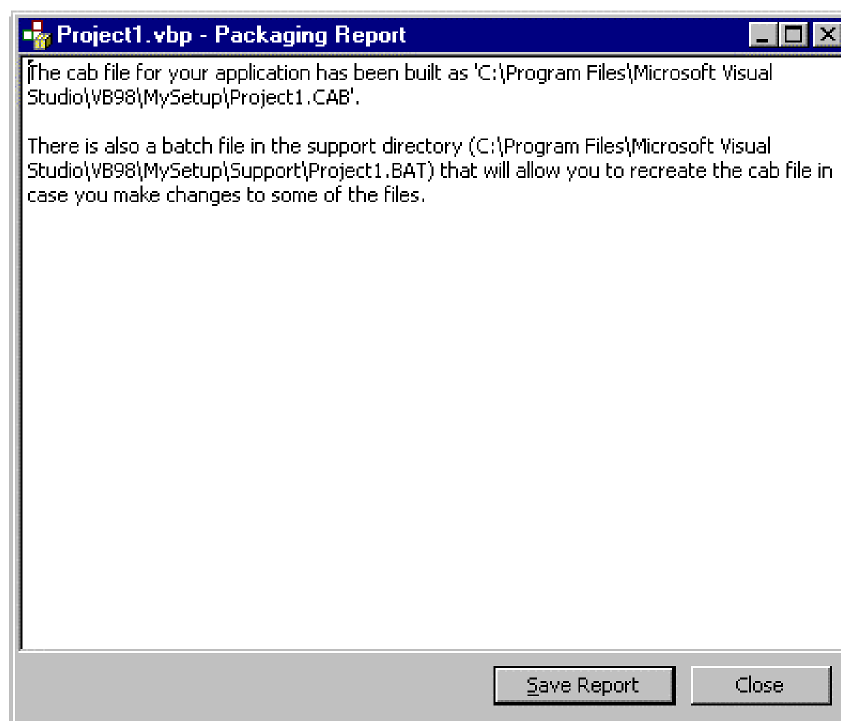
שלב זה מסיים את תהליך הכנת ערכת ההתקנה (תרשים 21.17).



תרשים 21.17

כדי לסיים את התהליך לחץ על לחצן **Finish**. לחיצה זו גורמת לתוכנה לכווץ את כל הקבצים הכלולים בערכה.

בסיום התהליך יציג האשף את החלון **Packaging Report** הכולל דוח קצר על תוכנית ההתקנה (תרשים 21.18). ניתן לשמור דוח זה כקובץ טקסט בלחיצה על הלחצן **Save Report**.



תרשים 21.18

בזה **תם** תהליך הכנת ערכת ההתקנה. ערכה זו נמסרת למשתמש (או הלקוח) והוא מתקין אותה ככל ערכת התקנה אחרת, בהפעלת הקובץ **Setup.exe**. במהלך ההתקנה ניתנת למשתמש האפשרות לבחור בתיקיה רצויה ולהתחיל בהתקנה. בסיומה, כל סביבת העבודה הנדרשת ליישום מותקנת במחשב. בנוסף נוצר בתפריט **תוכניות** שבתפריט **התחל** (את המסלול המדויק קבעת בערכת ההתקנה - בשלב ט') קיצור דרך אל קובץ ההפעלה של היישום, אשר כולל את שם היישום וסמלו.

נספח: התקליטור המצורף

מה בתקליטור?

- **גרסת הלימוד של שפת הפיתוח Visual Basic 6 - Visual Basic 6 Working Model**
- **קטעי קוד המקור שבספר.**
- מספר תוכנות עזר שימושיות.



הערה!

אם מנהל התקן כונן התקליטורים המותקן הוא 16 סיביות - ייתכן שתראה רק 8 תווים ראשונים של שם קובץ (במידה שהמקור ארוך יותר).
הסיבה: כונני תקליטורים במהירות x4 עובדים עם מנהל התקן שעבד בסביבת DOS ו-Windows 3.11 ויכול לעבוד גם עם Windows 95, למעט היכולת לזהות קבצים עם שמות ארוכים.
הפתרון: להתקין מנהל התקן 32 סיביות (אם קיים), או לקנות כונן תקליטורים חדש ולוודא שמצורף אליו מנהל התקן 32 סיביות.

אם אינך בטוח כיצד להתקין את התוכניות שבתקליטור פנה לאיש מקצוע.

קרא את קובץ ONCD שבתקליטור כדי לקבל עוד מידע לגבי התקליטור

תקליטור זה מופעל באופן אוטומטי עם הכנסתו לכונן.

אם התקליטור אינו מופעל באופן אוטומטי:

1. ודא שהתקליטור בכונן
2. בשולחן העבודה לחץ לחיצה כפולה על סמל **המחשב שלי**
3. לחץ לחיצה כפולה על סמל כונן התקליטורים
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**

קרא קובץ ONCD שנמצא בתקליטור לגבי תכולת התקליטור ולהוראות התקנה של חלק מהתוכנות המצורפות.

התיקיות הרלוונטיות לספר

התיקיות הרלוונטיות לספר זה הן:

Books\59214 שמכילה את קבצי קוד המקור הנמצאים בספר

Software\Visual Basic 6 Working Model שמכילה תוכנת VB

היכן נמצאות התוכניות של ספר זה?

התיקיה הרלוונטית לספר זה: **59214**

בתיקיה **X:\Books\59214** נמצאים רוב קטעי קוד המקור שמופיעים בספר.

קוד המקור נמצא בקבצי טקסט (txt). תוכל להשתמש בקבצים אלה כדי לבצע פעולת העתק והדבק לתוך חלון הקוד שבתוכנת Visual Basic.

בתיקיות אחרות תחת התיקיה Books נמצאים קבצים הרלוונטיים לספרים אחרים של ההוצאה.

העתקת קבצי המקור לדיסק

לכל ספר יש מספר ייחודי, הבנוי מה-דאנאקוד, המופיע על העטיפה האחורית של הספר, והוא מתחיל בספרות 59 (לדוגמה 59315).

מספר הפריט של ספר זה הוא: **59214**

כדי להתקין את קוד המקור לחץ על **קוד מקור** וציין את מספר התיקיה **59214.**

התקנת קוד המקור פירושה **העתקה** (לא התקנה) של הקבצים מהתקליטור לדיסק הקשיח במחשב באישי שלך.

לאחר שיועתק קוד המקור הוא יימצא בדיסק C בתיקיה HodAmiBooks\59214.

כדי להריץ תוכניות עליך לוודא שבמחשב מותקנת תוכנת Visual Basic.

התקנת Visual Basic 6 Working Model

תוכנה זו הינה גירסה מיוחדת שהוציאה מיקרוסופט. היא אינה מלאה אך שימושית בהחלט. תוכל להריץ כמעט את כל התוכניות שבספר. מיקרוסופט והוד-עמי אינן תומכות בתוכנה.

התקנת Visual Basic 6 Working Model דורשת כי דפדפן האינטרנט Internet Explorer יהיה מותקן במחשב (בגירסה 4 ומעלה). לצורך התקנת **Visual Basic 6 Working Model** עליך **להסיר** את התקנת Internet Explorer גירסה 3, אם קיימת, ורק אז לבצע את התקנת Visual Basic 6 Working Model.

לתשומת לבך: במידה ובמחשב היתה מותקנת גירסה קודמת של Internet Explorer צפויה במהלך ההתקנה להופיע תיבת דו-שיח המורה על **התנגשות בין גירסאות**. לשאלה **האם ברצונך לשמור את הקובץ הקיים במחשבך** השב **כן** וההתקנה תמשיך כרגיל.

התוכנה ניתנת כבונוס ללא תשלום לרוכשי הספר

הוצאת הוד-עמי אינה מספקת תמיכה במוצר.

בדוק שמותקן דפדפן Internet Explorer במחשב לפני התקנת המהדר.

התקנת Visual Basic 6 Working Model מתבצעת באופן הבא:

1. ודא שהתקליטור בכונן
2. בשולחן העבודה לחץ לחיצה כפולה על סמל **המחשב שלי**
3. לחץ לחיצה כפולה על סמל כונן התקליטורים
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**.
5. לחץ על **התקנת תוכנות**.
6. לחץ על לחצן האפשרויות ליד **אני מאשר את תנאי השימוש בתקליטור**.
7. לחץ **המשך**.
8. פתח את התיקיה **Visual Basic 6 Working Model**.
9. בחר בקובץ **Setup.exe** ולחץ על **פתח**.
10. בתיבת הדו-שיח **הפעלה** לחץ על **אישור** וההתקנה תחל.
11. במסך הראשון: **Visual Basic 6 Working Model** לחץ על **Next**.

12. במסך השני: **End User License Agreement** בחר: **I accept the agreement** ולחץ על **Next**.

13. במסך השלישי: **Product Number and user ID** הקלד את שמך ולחץ על **Next**.

14. במסך **Install DCOM98** ודא שתיבת הסימון מסומנת ולחץ על **Next**.

15. לאחר התקנת הרכיב **DCOM98** תוכנית ההתקנה תודיע שכעת יש לאתחל את המחשב מחדש. לחץ על **OK**.

16. לאחר שהמחשב יעלה מחדש, תימשך ההתקנה. המסך הבא שיופיע הוא **Choose Common Install Folder**. בדוק אם המיקום והנתיב שמציעה תוכנית ההתקנה מתאימים לך. אם לא, לחץ על **Browse** ובחר מיקום אחר. כעת כשהמיקום והנתיב מתאימים, לחץ על **Next**.

17. במסך: **Visual Basic 6 Working Model Setup** תופיע הודעה שיש לסגור כל יישום פתוח (למעט תוכנית ההתקנה), עשה זאת (באמצעות לחיצה על לחצן היישום שבשורת המשימות) ולחץ על **Continue**.

18. בשלב הבא לחץ על **OK** והמתן בעת שתוכנית ההתקנה מחפשת רכיבים מותקנים.

19. במסך הבא שיופיע בחר בסוג ההתקנה. אם יש לך מספיק מקום בדיסק שבחרת בחר בהתקנת **Typical**.

20. אם תוכנית ההתקנה תבקש להחליף Dll ענה **No** ואחר כשתשאל אם ברצונך להשאיר את הקבצים כמות שהם ענה **Yes**.

21. בשלב זה תתבקש לאתחל את Windows מחדש, לחץ על **Restart Windows**.

22. כשהמחשב "יעלה" מחדש תופיע תיבת דו-שיח המאפשרת רישום עותק התוכנה באמצעות האינטרנט. פעל לפי ראות עיניך.

ספר זה מבוסס על גרסאות Enterprise ו- Professional של Visual Basic 6 ומתאים לעבודה גם בגרסת Standard, למעט גרסת Working Model (קוד המקור לא נבדק בגירסה זו).

קטלוג ספרים

כדי לצפות בקטלוג העדכני ביותר ולצפות במבצעים המיוחדים מומלץ לבקר באתר האינטרנט של ההוצאה בכתובת **www.hod-ami.co.il**. עיון בקטלוג ספרי המחשבים של הוצאת הוד-עמי כולל:

- תוכן עניינים מלא של הספר
- פרק לדוגמה
- מגה-אינדקס לחלק מהספרים

את כל המידע ניתן לראות וגם להדפיס. לצורך זה יש להתקין את תוכנת Acrobat Reader מהתפריט **התקנת תוכנות** (ראה התקנה בהמשך).

הקטלוג מומלץ לצפיה ברזולוציית מסך של 600x800.

הקטלוג מומלץ לצפייה באמצעות Internet Explorer 5 ומעלה. הדפדפן מצורף בתקליטור. כדי להתקין אותו בחר בתפריט **התקנת תוכנות** ובחר באפשרות **תוכנות נוספות** (ראה התקנה בהמשך).

Acrobat Reader - התקנה

1. ודא שהתקליטור נמצא בכונן התקליטורים
2. בשולחן העבודה לחץ לחיצה כפולה על הסמל **המחשב שלי**
3. לחץ לחיצה כפולה על סמל כונן התקליטורים
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**
5. לחץ על **התקנת תוכנות**
6. לחץ על לחצן האפשרויות ליד **אני מאשר את תנאי השימוש בתקליטור**
7. לחץ **המשך**
8. לחץ על **Acrobat Reader**
9. אשף ההתקנה מתקין את הרכיבים הנדרשים.

   במהלך ההתקנה עליך ללחוץ על **Next**, **Accept** ו-**Next** פעם נוספת כדי לבצע את ההתקנה.
10. בסיום ההתקנה עשויה להופיע על המסך תיבת דו-שיח **התנגשות בין גירסאות** ומייד אחר כך להיעלם. על המסך תופיע תיבת הודעה של תוכנית ההתקנה. לחץ על **אישור.** בתיבת הדו-שיח **התנגשות בין גירסאות** לחץ על **כן**, כדי לשמור את גרסת הקובץ שלך.

מה עוד בתקליטור?

הוצאת **הוד-עמי** מפיצה תוכנות אלו כבונוס ללקוחות ההוצאה, ואינה מתיימרת לגבות תשלום עבור התוכניות המצורפות ו/או לתמוך בהם.



אזהרה:

השימוש בתקליטור זה הוא על אחריותו הבלעדית של המשתמש. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם. הוצאת **הוד-עמי** אינה אחראית, בכל צורה שהיא, לאופן ולטיב התוכנות המותקנות.

בכל שאלה לגבי תוכנה הנמצאת בתקליטור, יש לפנות למפתחי התוכנה (כל תוכנה בנפרד) כפי שמצוין בקבצי העזרה של התוכנה המדוברת.

הקבצים הם גרסאות **שיתופיות** (ShareWare) **וחופשיות** (FreeWare).

גרסת ShareWare מאפשרת לך, המשתמש, לבדוק את יעילות התוכנה ואת תאימותה לעבודה אותה אתה מבצע. אם נמצאה התוכנה מתאימה לצרכיך, עליך לשלם למפתחיה תשלום סמלי (לפי הרשום בקבצי העזרה של כל תוכנה ותוכנה בנפרד) כדי לקבל רישיון מלא לשימוש בה. קבלת רישיון לשימוש בתוכנה יפתח בפניך מיגוון אפשרויות שלא עמדו לרשותך בהפעלת גרסת ה-ShareWare.

התקנת תוכנת גלישה לאינטרנט
Microsoft Internet Explorer 6

גירסה זו של Internet Explorer המצורפת בתקליטור, מיועדת להתקנה אך ורק במערכות בהן מערכת ההפעלה היא Windows 98 ומעלה ו<u>בעברית מלאה</u> (לחצן "התחל" בפינה הימנית התחתונה של המסך).

אין אפשרות להתקין גירסה זו במחשב בו מותקנת מערכת ההפעלה Windows 95.

אם תנסה להתקין גירסה זו במחשב בו מערכת ההפעלה אינה עברית אתה עלול לגרום נזק למערכת ההפעלה, דבר שעשוי לדרוש ממך התקנה מחדש של מערכת ההפעלה והתוכנות הנלוות.

אם יצאת מממשק התקליטור:

1. ודא שהתקליטור נמצא בכונן התקליטורים.
2. בשולחן העבודה, לחץ לחיצה כפולה על הסמל **המחשב שלי**.
3. לחץ לחיצה כפולה על סמל **כונן התקליטורים**.
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**.
5. לחץ על **התקנת תוכנות**.
6. לחץ על לחצן האפשרויות ליד **אני מאשר את תנאי השימוש בתקליטור**.
7. לחץ **המשך**.
8. לחץ על **תוכנות נוספות**.
9. מצא את התיקיה **Software\IE6\i386** ובחר בקובץ בשם **Setup.exe**.
10. לחץ עליו לחיצה כפולה.
11. לחץ על לחצן **אישור**.
12. פעל על פי ההוראות שעל המסך.

FontsPekan

קובץ זה יתקין במחשב 2 גופנים בעברית לשימושכם. בסיום ההתקנה יש לבצע את הפעולות הבאות:

1. לחץ על **התחל**, הצבע על **הגדרות**, ובחר **בלוח הבקרה**.
2. לחץ לחיצה כפולה על הסמל **גופנים**.
3. פתח את תפריט **קובץ** ובחר באפשרות **התקנת גופן חדש**.
4. עבור לתיקיה **C:\FontsPekan**.
5. לחץ על לחצן **בחר הכל** (סה"כ יש בתיקיה 2 גופנים).
6. ודא שתיבת הסימון **העתק גופנים לתיקיית הגופנים** מסומנת.
7. לחץ **אישור**.
8. סגור את חלון התיקיה **Fonts**.
9. סגור את חלון **לוח הבקרה**.

כעת, מוכנים הגופנים לשימוש בכל התוכנות המותקנות במחשב שלך: Word, Excel, PowerPoint וגם בתוכנות גרפיות, כגון Paint Shop Pro ו-PhotoShop.

הגופנים נקראים Tml-JUMP ו-Tml-step ויופיעו בתחתית רשימת שמות הגופנים (בדרך כלל). הרי דוגמה שלהם:

Tml-step

אבגדהוזחטיכךלמםנןסעפףצץקרשת1234567890!

Tml-JUMP

אבגדהוזחטיכךלמםנןסעפףצץקרשת1234567890

תיקיה ראשית SoftWare
(רשימה חלקית ועשויה להשתנות)

שם תוכנה	תיאור	קובץ הפעלה	מבצע
Adobe Acrobat	תוכנה לצפייה בקבצי pdf, כולל תמיכה בעברית	ARME505Heb.exe	התקנה
ICQ	תוכנה לתקשורת אישית באינטרנט	ICQPro-2003a-3800-icqpro2003a.exe	התקנה
IEPowerToys	תוכניות שירות עבור Internet Explorer	IEPowerToys.exe	פריסה
WinAmp	תוכנה להשמעת קבצי MP3 (מוסיקה)	WinAmp3_0-full.exe	התקנה
WinZip	תוכנית לפריסה/דחיסה של קבצים	WinZip81sr1.exe	התקנה

שאלות נפוצות

בחלק זה של התקליטור תמצא אוסף של שאלות נפוצות ותשובות במיגוון נושאים: איך מחפשים בספר? כיצד מזמינים ספר דרך האתר באינטרנט? למי פונים בבקשת תמיכה? האם ניתן לבצע הזמנה טלפונית? וכדומה.

הערה חשובה!

הוצאת הוד-עמי אינה תומכת בתוכנות המסופקות בתקליטור זה. התוכנות ניתנות ללקוח כבונוס, לשימושו האישי, והן מסופקות AS-IS כפי שסופקו להוצאת הוד-עמי על ידי יצרן התוכנה.

התמיכה ניתנת ללקוחות ההוצאה בנוגע לספרים הרואים אור בהוצאה.

ניתן לשאול שאלות, להציע הצעות ולבקש עזרה בהפעלת קוד או קובץ המוזכר בספר.

מחלקת התמיכה של הוצאת הוד-עמי אינה תומכת ביישומים ובתוכנות המסופקות בתקליטור.

צור קשר

תוכל ליצור עמנו קשר באמצעי התקשורת הבאים:

טלפון: 09-9564716

פקס: 09-9571582

דואר אלקטרוני בנושא כללי: info@hod-ami.co.il

דואר אלקטרוני בנושאי תמיכה: support@hod-ami.co.il

אנשי התמיכה שלנו ערוכים לתת לך את העזרה הדרושה לך בשימוש בספר.

אין אנו נותנים תמיכה הקשורה בתוכנות המצורפות בתקליטור.

בפנייתך אלינו לקבלת תמיכה, אנא הצטייד בנתונים הבאים:

1. שמו המלא של הספר
2. מספר דאנאקוד, כפי שמופיע על הכריכה האחורית של הספר (למשל בספר זה 259-10214)
3. פרטים מדויקים של הבעיה (הכוללים מספר פרק, מספר עמוד, שם קובץ)
4. אמצעי קשר בו נוכל להשיג אותך: טלפון נייד, פקס, E-Mail.

אם יש לך שאלה או הערה בנוגע לספר מספרי הוצאת הוד-עמי, נשמח לשמוע ממך. יש מספר דרכים בהן תוכל להעביר לנו את הערותיך:

בדואר אלקטרוני: support@hod-ami.co.il

בפקס: 09-9571582

בדואר: ת.ד. 6108 הרצליה 46160

בפנייתך אנא ציין את הפרטים הבאים: שמו המדויק של הספר, מספר עמוד, מספר שורה מלמעלה/מלמטה, מה לא בסדר, מה לדעתך צריך להיות.

תודה רבה שבחרתם הוד-עמי

טלפון: 09-9564716

פקס: 09-9571582

אתר באינטרנט: http://www.hod-ami.co.il

דואר אלקטרוני: info@hod-ami.co.il

פנה לקובץ ONCD.PDF לפרטים נוספים על תכולת התקליטור.

אינדקס

א

ח

ט

ל

מ

ק

ש

ת